https://algs4.cs.princeton.edu

4.3 MINIMUM SPANNING TREES

- *introduction*
- *cut property*
- *edge-weighted graph API*
- *Kruskal's algorithm*
- *Prim's algorithm*

https://algs4.cs.princeton.edu

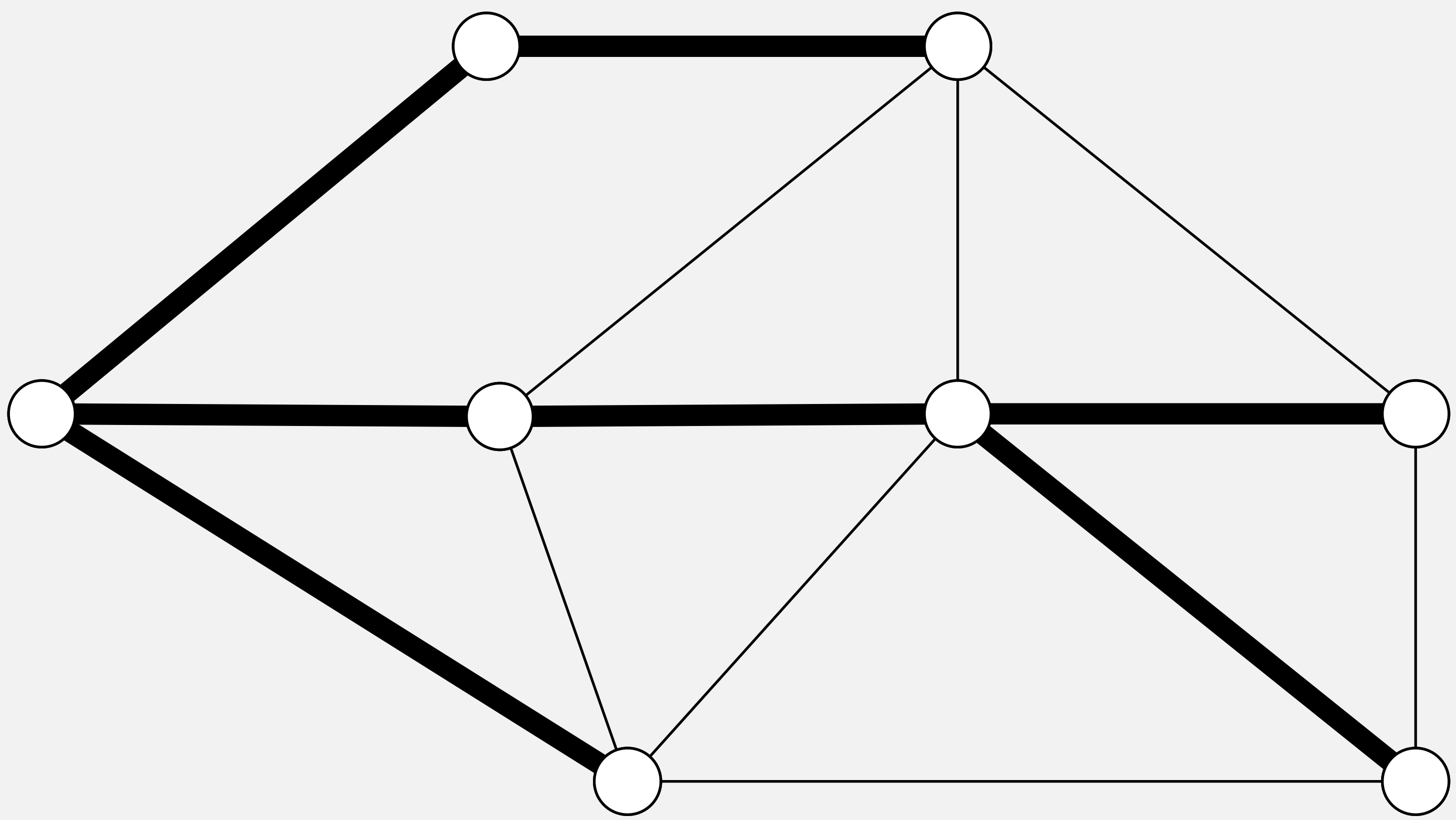
4.3 Minimum Spanning Trees

- *introduction*
- *cut property*
- *edge-weighted graph API*
- *Kruskal's algorithm*
- *Prim's algorithm*

Spanning tree

Def. A spanning tree of G is a subgraph T that is:

- A tree: connected and acyclic.
- Spanning: includes all of the vertices.

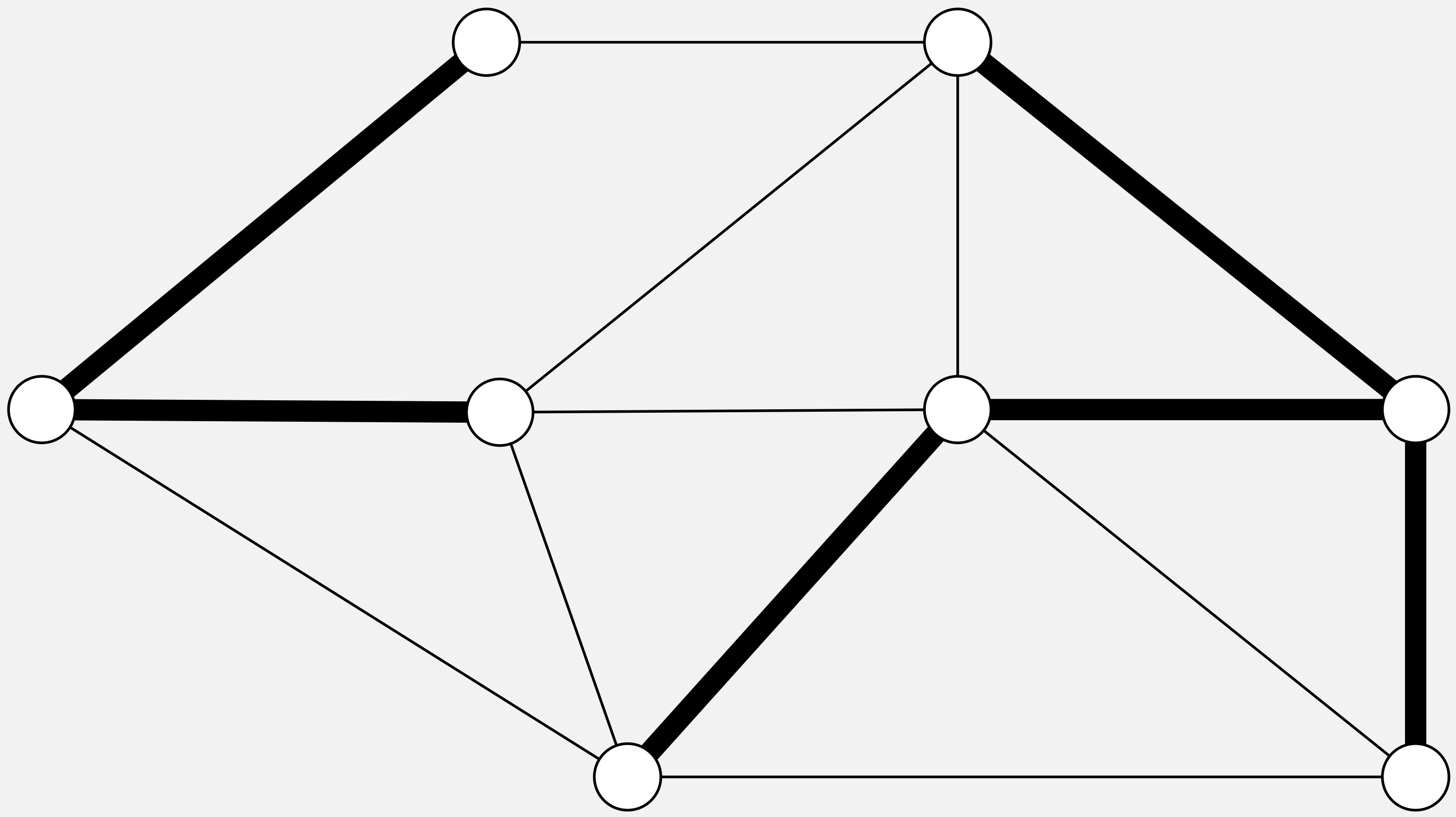


graph G
spanning tree T

Spanning tree

Def. A spanning tree of G is a subgraph T that is:

- A tree: connected and acyclic.
- Spanning: includes all of the vertices.

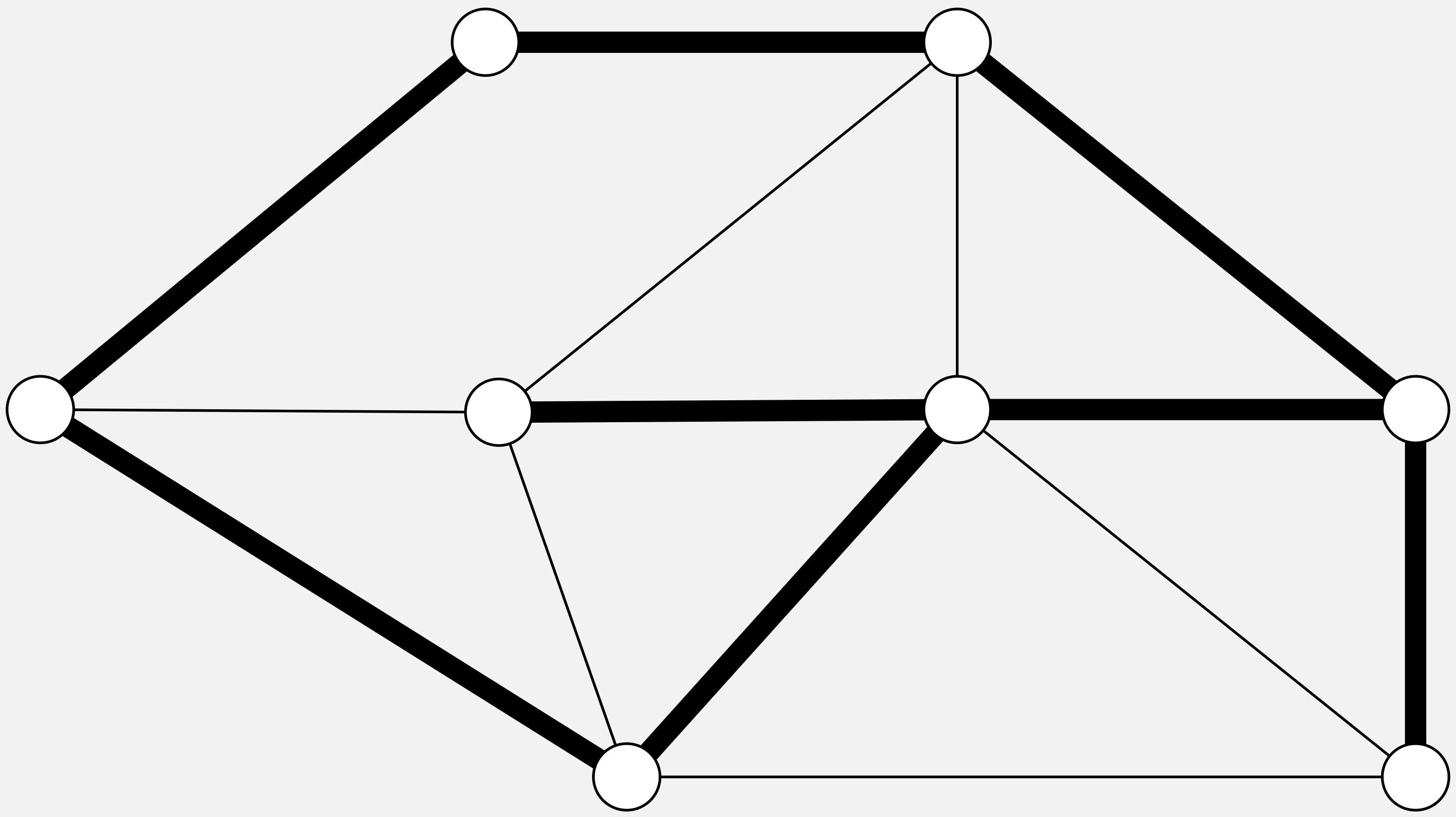


not connected

Spanning tree

Def. A spanning tree of G is a subgraph T that is:

- A tree: connected and acyclic.
- Spanning: includes all of the vertices.

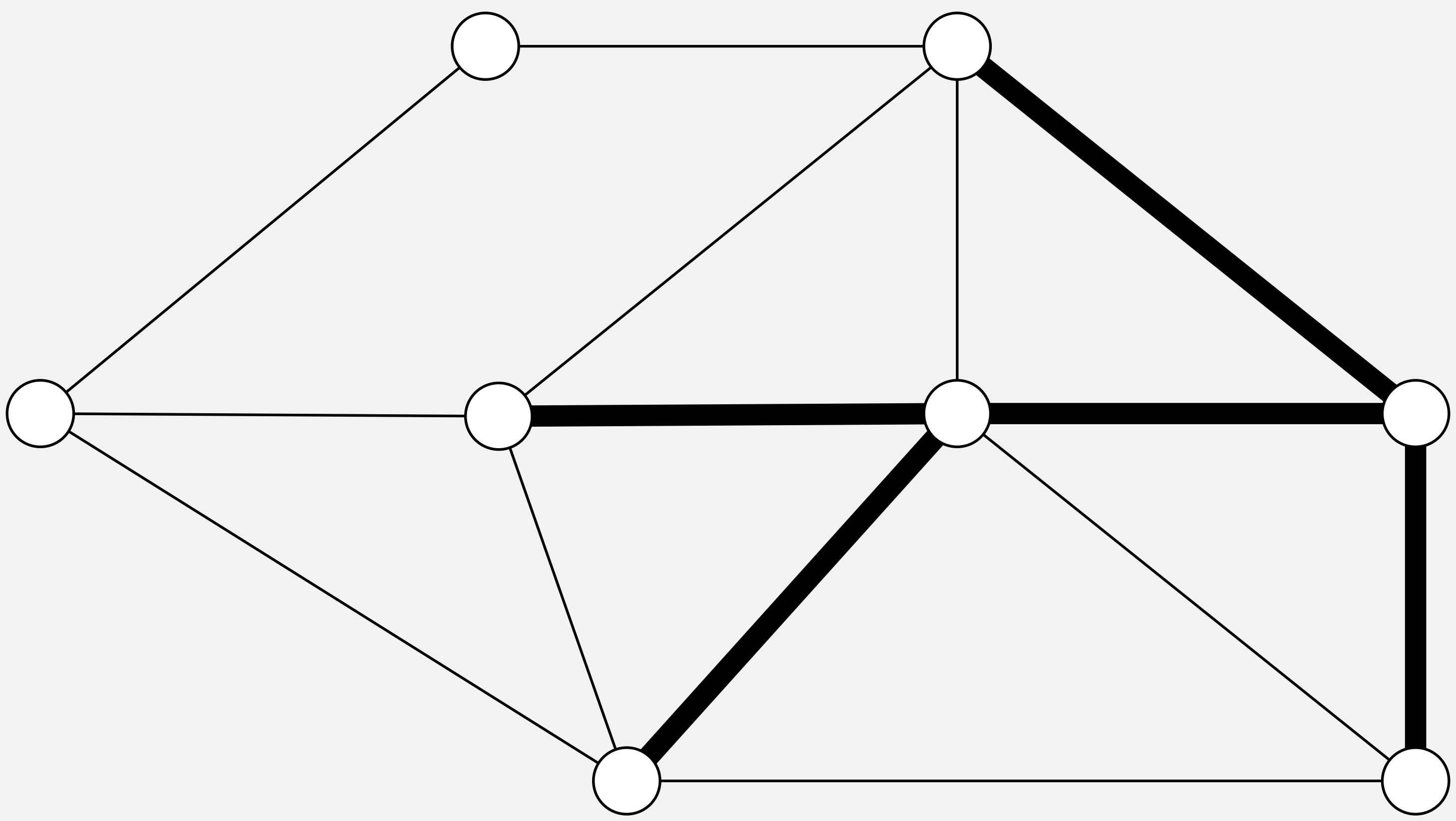


not a tree (cyclic)

Spanning tree

Def. A spanning tree of G is a subgraph T that is:

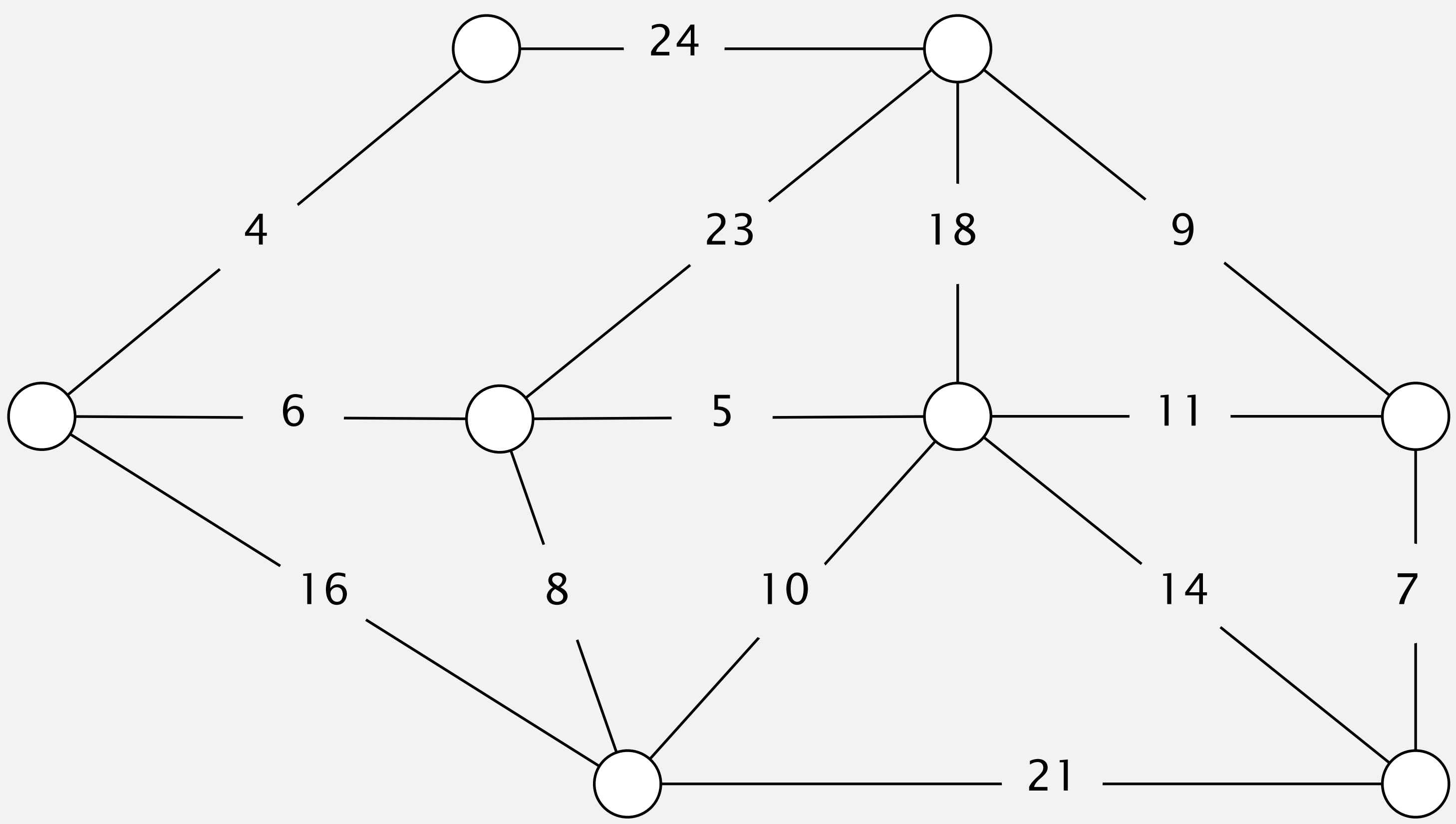
- A tree: connected and acyclic.
- Spanning: includes all of the vertices.



not spanning

Minimum spanning tree problem

Input. Connected, undirected graph G with positive edge weights.

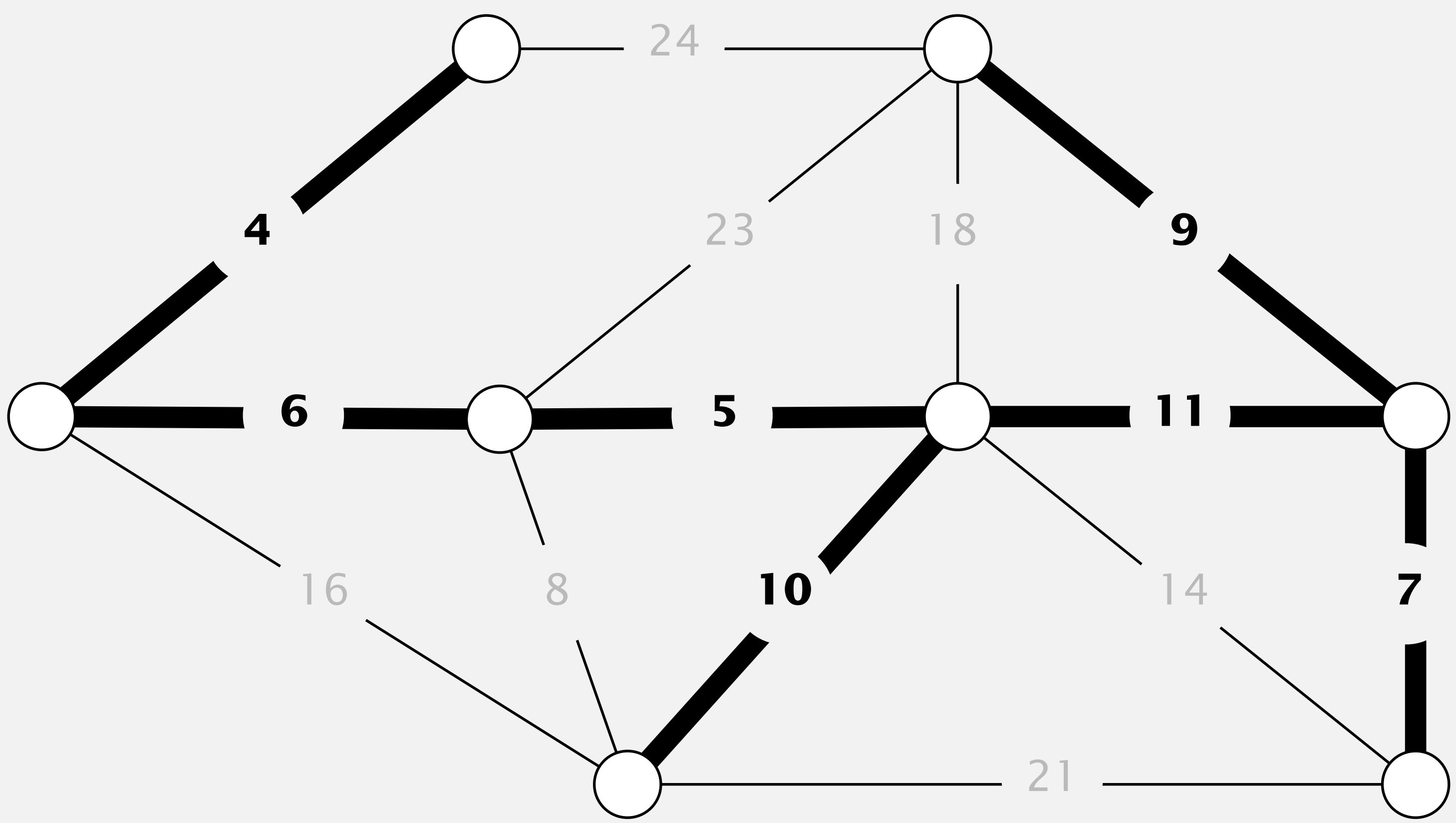


edge-weighted graph G

Minimum spanning tree problem

Input. Connected, undirected graph G with positive edge weights.

Output. A spanning tree of minimum weight.



minimum spanning tree T
(weight = 52 = 4 + 6 + 10 + 5 + 11 + 9 + 7)

Brute force. Try all spanning trees?

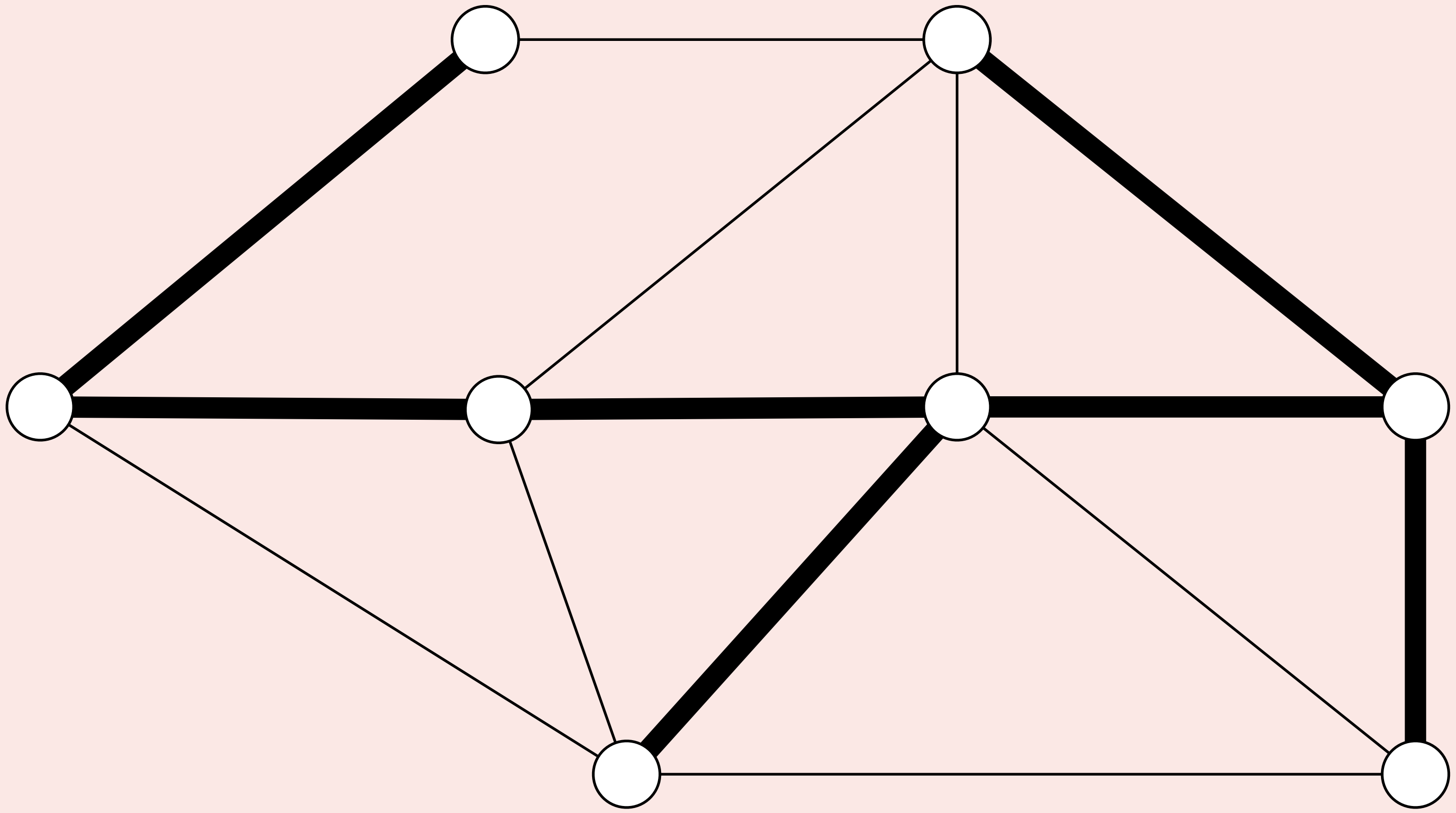
Let T be a spanning tree of a connected graph G with V vertices.
Which of the following statements are true?

A. T contains exactly $V - 1$ edges.

B. Removing any edge from T disconnects it.

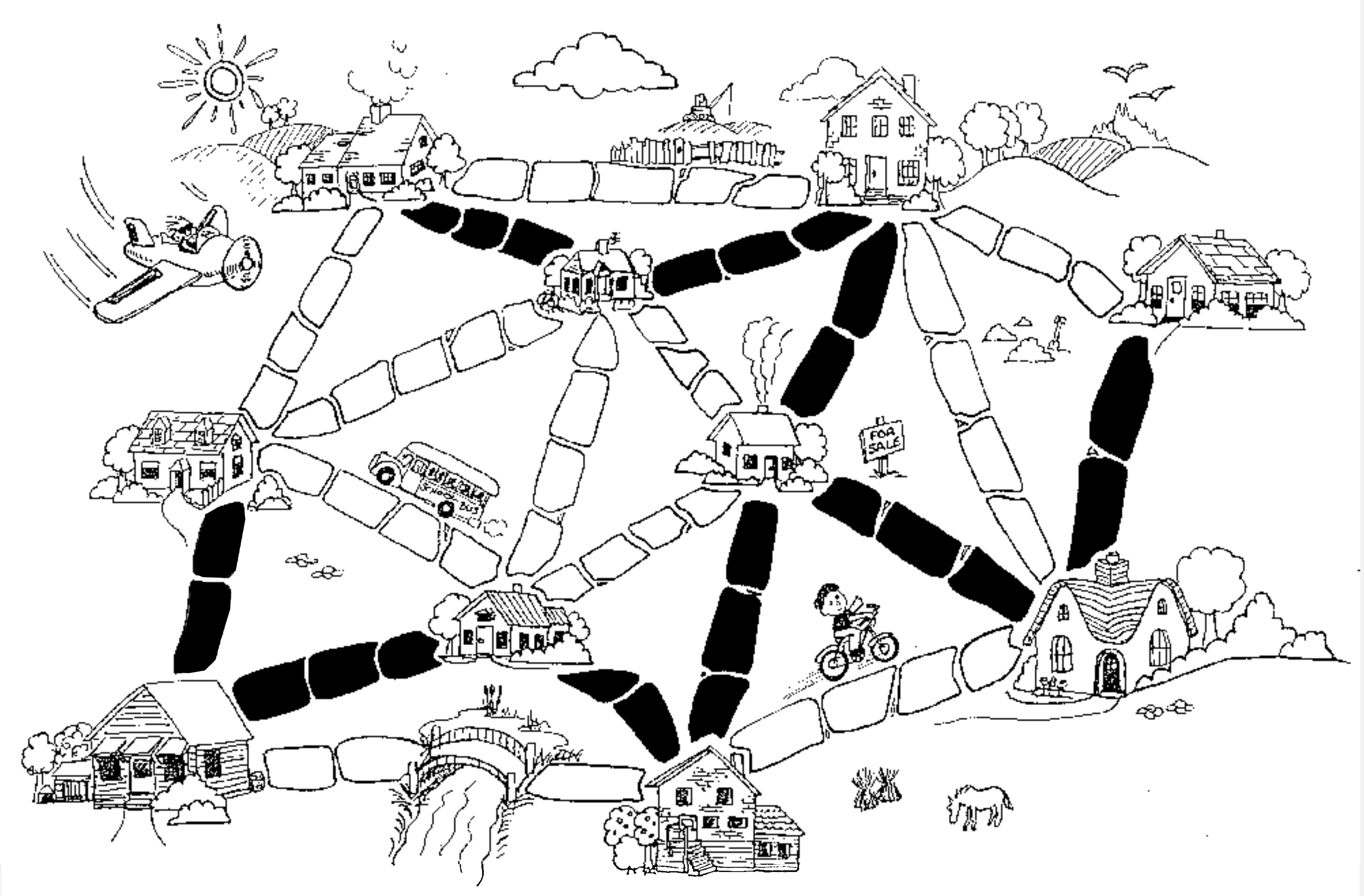
C. Adding any edge to T creates a cycle.

D. All of the above.



spanning tree T of graph G

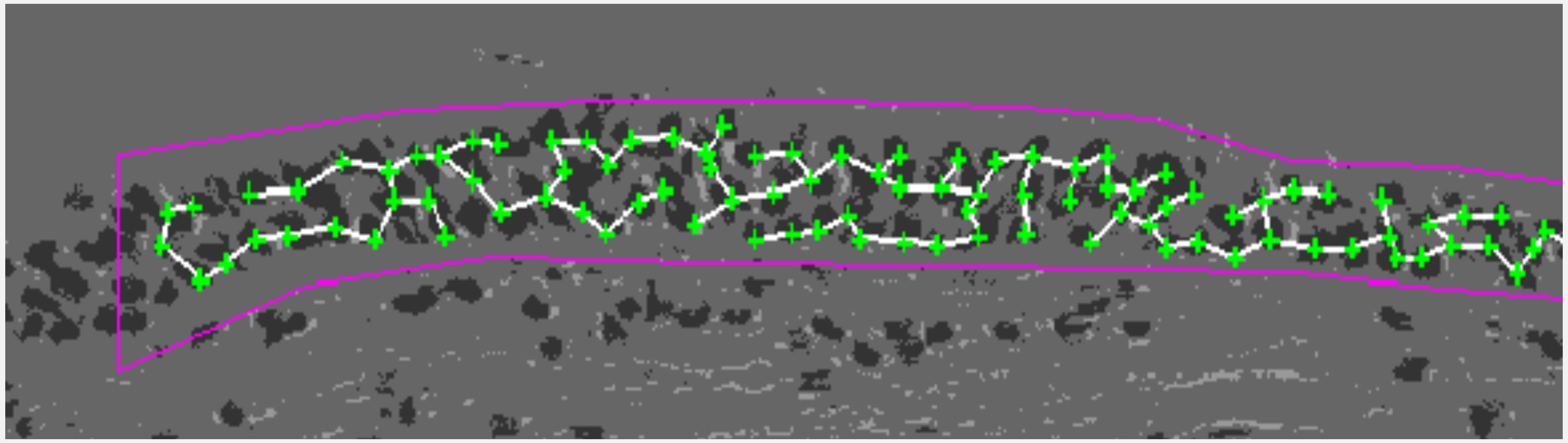
Network design

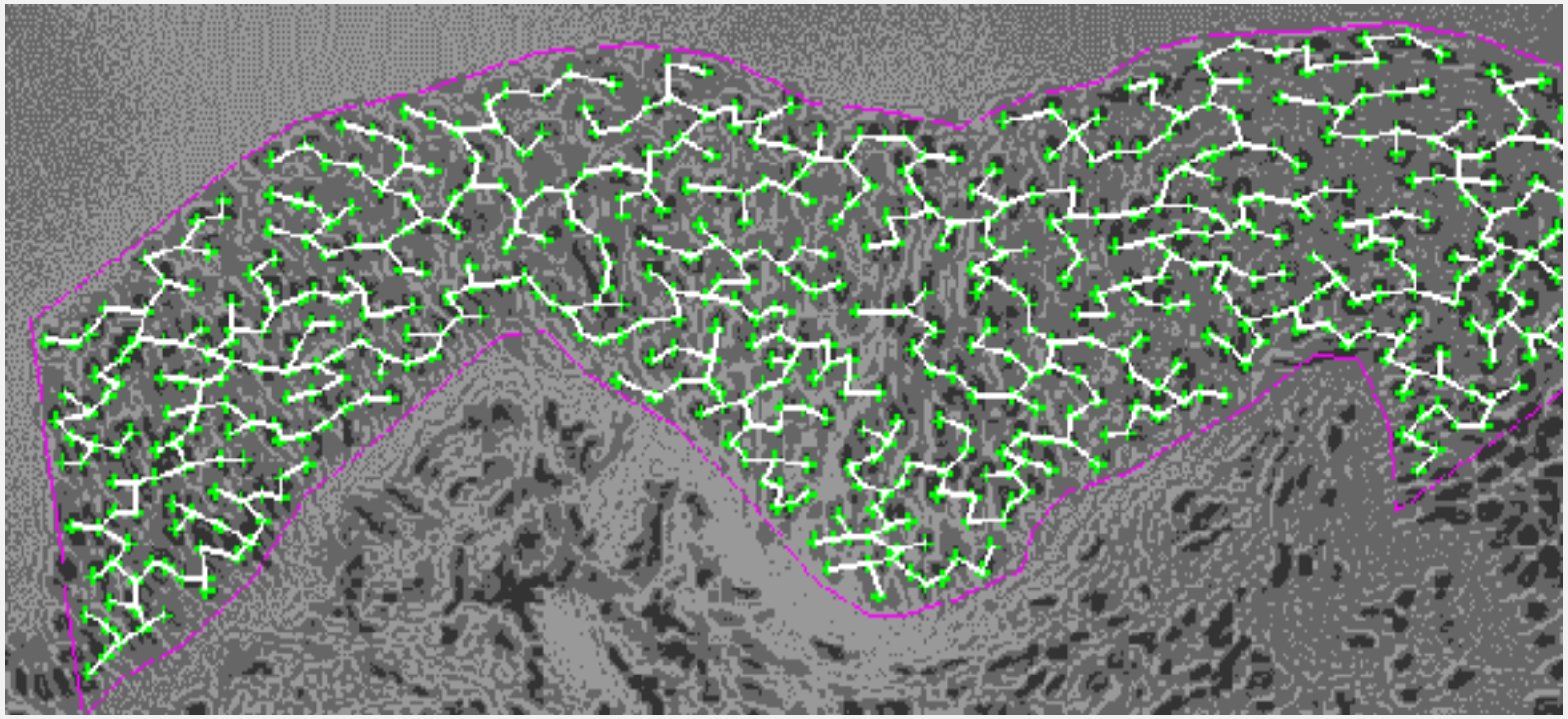


http://www.utdallas.edu/~besp/teaching/mst-applications.pdf

Medical image processing

MST describes arrangement of nuclei in the epithelium for cancer research



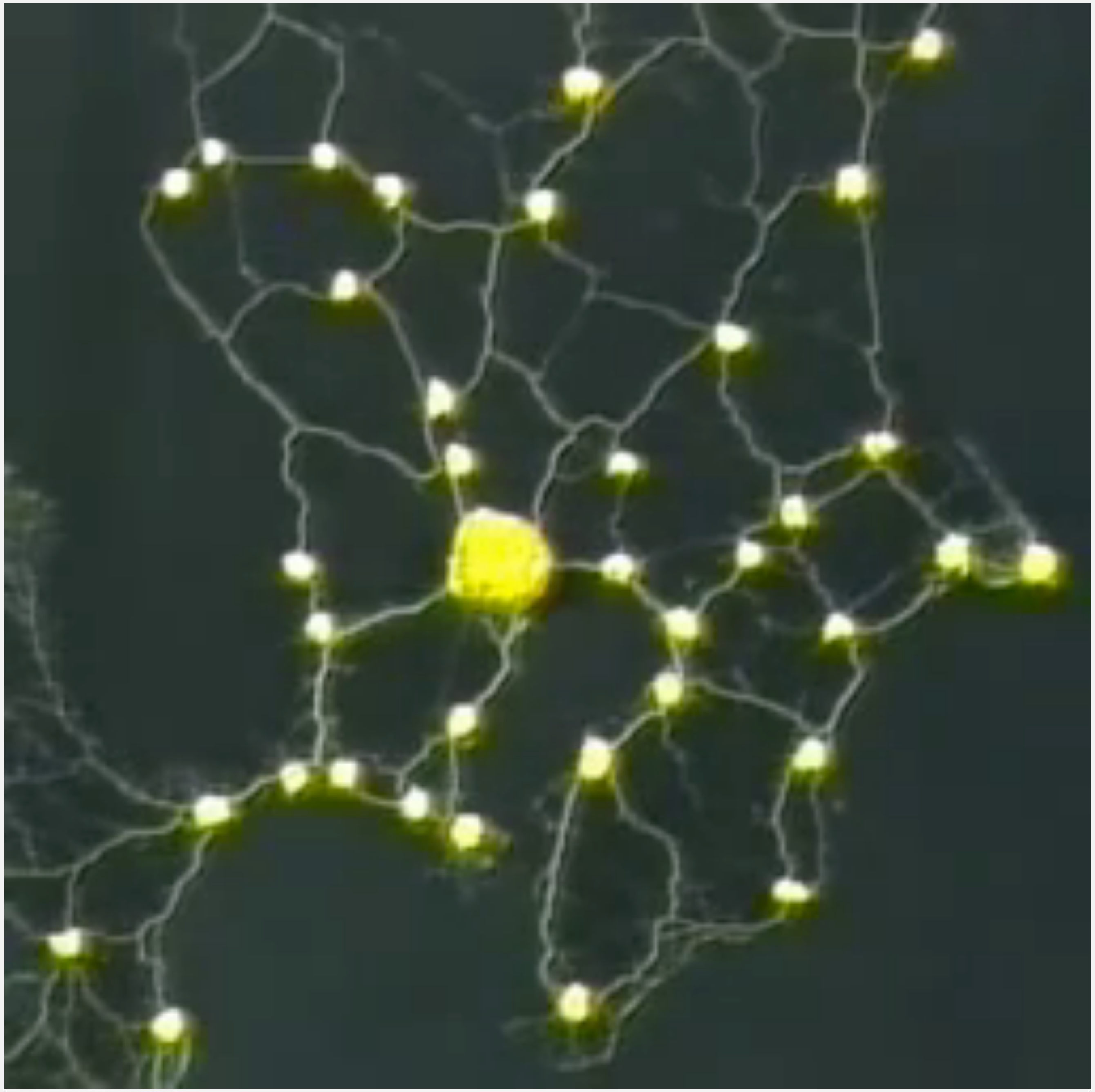


http://www.bccrc.ca/ci/ta01_archlevel.html

Slime mold grows network just like Tokyo rail system

Rules for Biologically Inspired Adaptive Network Design

Atsushi Tero,[1,2] Seiji Takagi,[1] Tetsu Saigusa,[3] Kentaro Ito,[1] Dan P. Bebber,[4] Mark D. Fricker,[4] Kenji Yumiki,[5] Ryo Kobayashi,[5,6] Toshiyuki Nakagaki[1,6]*



https://www.youtube.com/watch?v=GwKuFREOgmo

Applications

MST is fundamental problem with diverse applications.

- Cluster analysis.
- Real-time face verification.
- LDPC codes for error correction.
- Image registration with Renyi entropy.
- Curvilinear feature extraction in computer vision.
- Find road networks in satellite and aerial imagery.
- Handwriting recognition of mathematical expressions.
- Measuring homogeneity of two-dimensional materials.
  Model locality of particle interactions in turbulent fluid flows.
- Reducing data storage in sequencing amino acids in a protein.
- Autoconfig protocol for Ethernet bridging to avoid cycles in a network.
- Network design (communication, electrical, hydraulic, computer, road).
- Approximation algorithms for **NP**-hard problems (e.g., TSP, Steiner tree).

http://www.ics.uci.edu/~eppstein/gina/mst.html
http://www.utdallas.edu/~besp/teaching/mst-applications.pdf

https://algs4.cs.princeton.edu

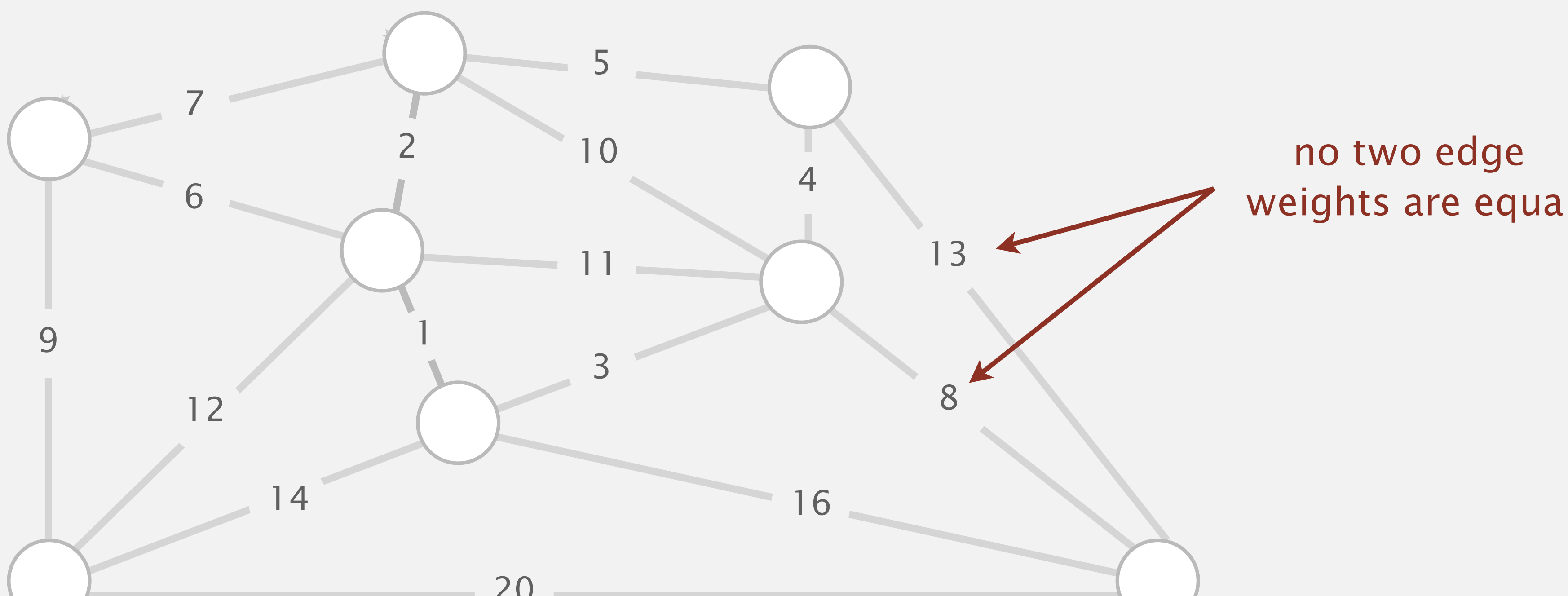
4.3 Minimum Spanning Trees

Simplifying assumptions

For simplicity, we assume:

- No parallel edges.
- The graph is connected. $\Rightarrow$ MST exists.
- The edge weights are distinct. $\Rightarrow$ MST is unique. ← see Exercise 4.3.3

Note. Algorithms still work even if parallel edges or duplicate edge weights.

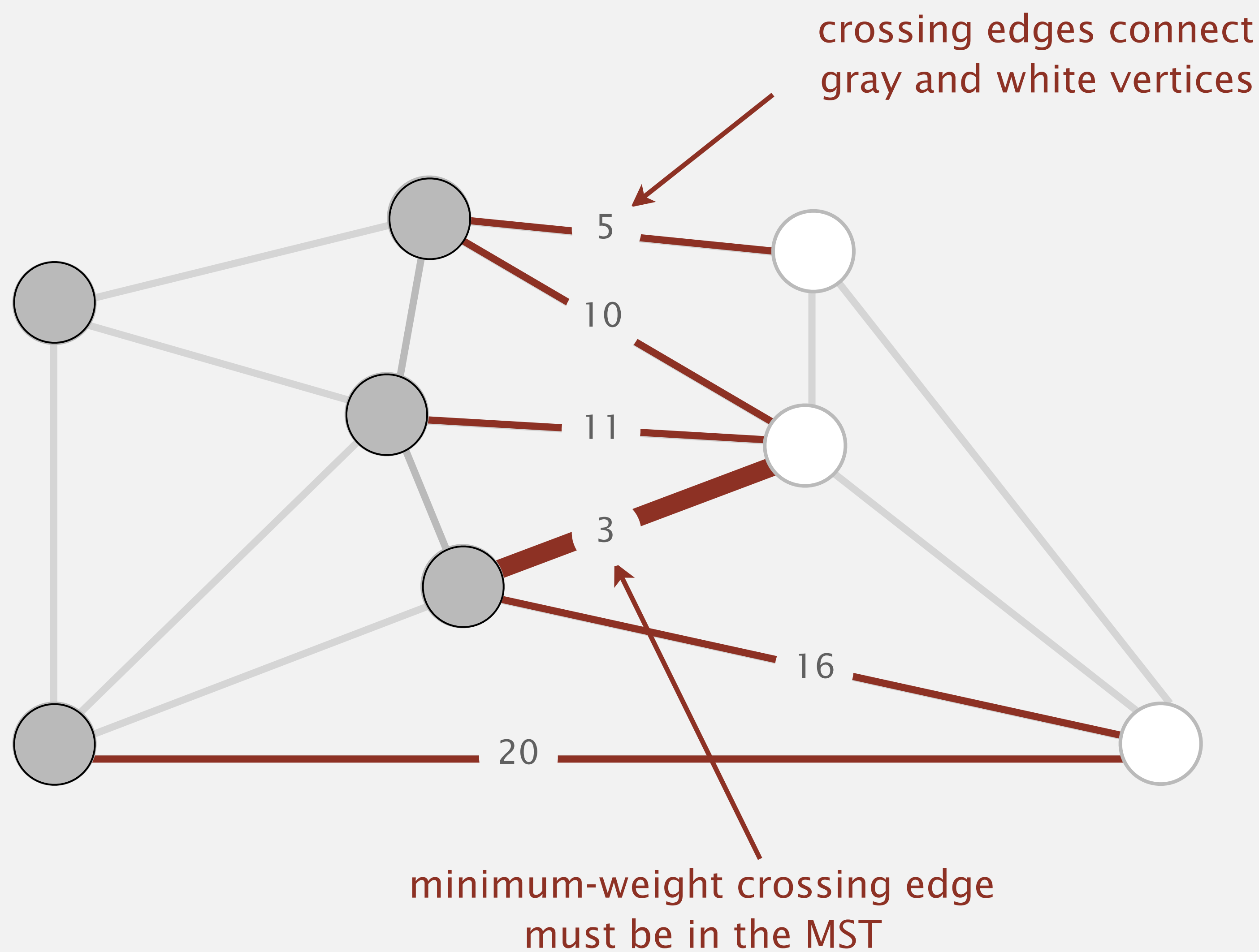


Cut property

Def. A cut in a graph is a partition of its vertices into two (nonempty) sets.

Def. A crossing edge connects a vertex in one set with a vertex in the other.

Cut property. Given any cut, the crossing edge of min weight is in the MST.



Minimum spanning trees: quiz 2

Which is the min weight edge crossing the cut $\{2, 3, 5, 6\}$?

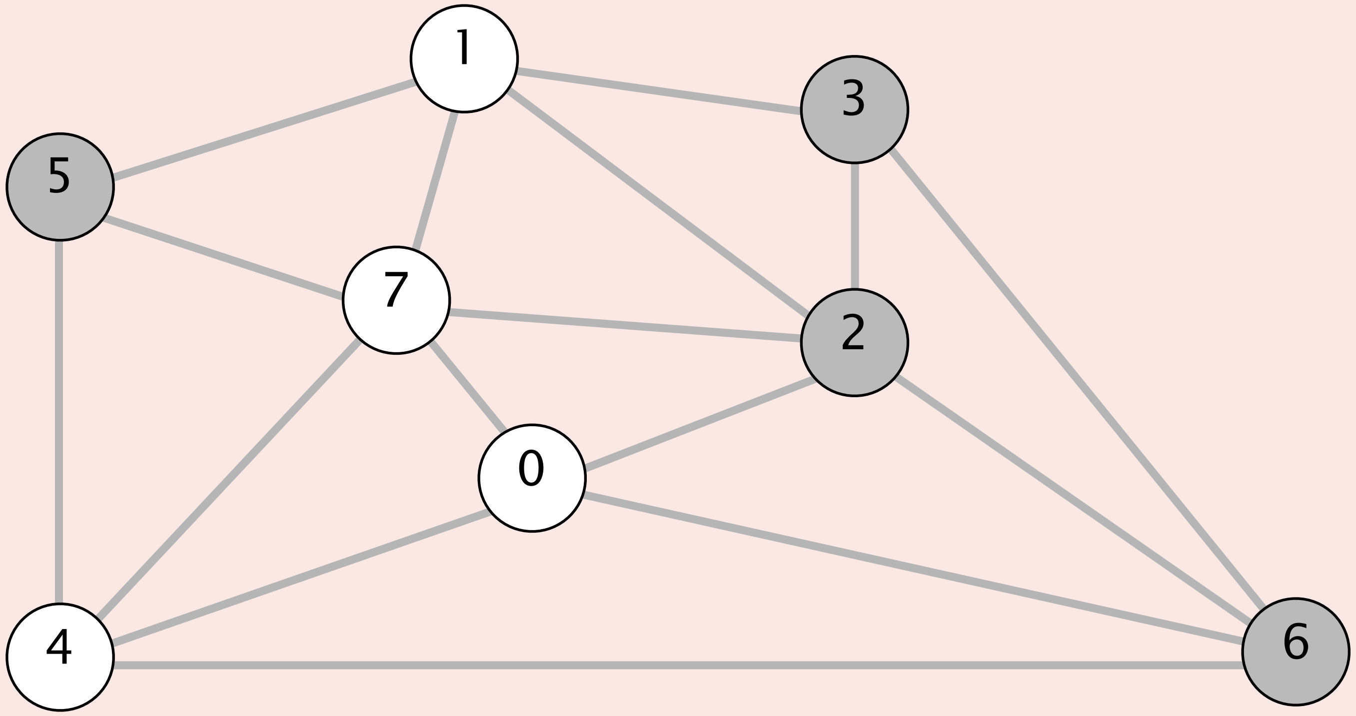
A. 0–7 (0.16)

B. 2–3 (0.17)

C. 0–2 (0.26)

D. 5–7 (0.28)

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93



Cut property: correctness proof

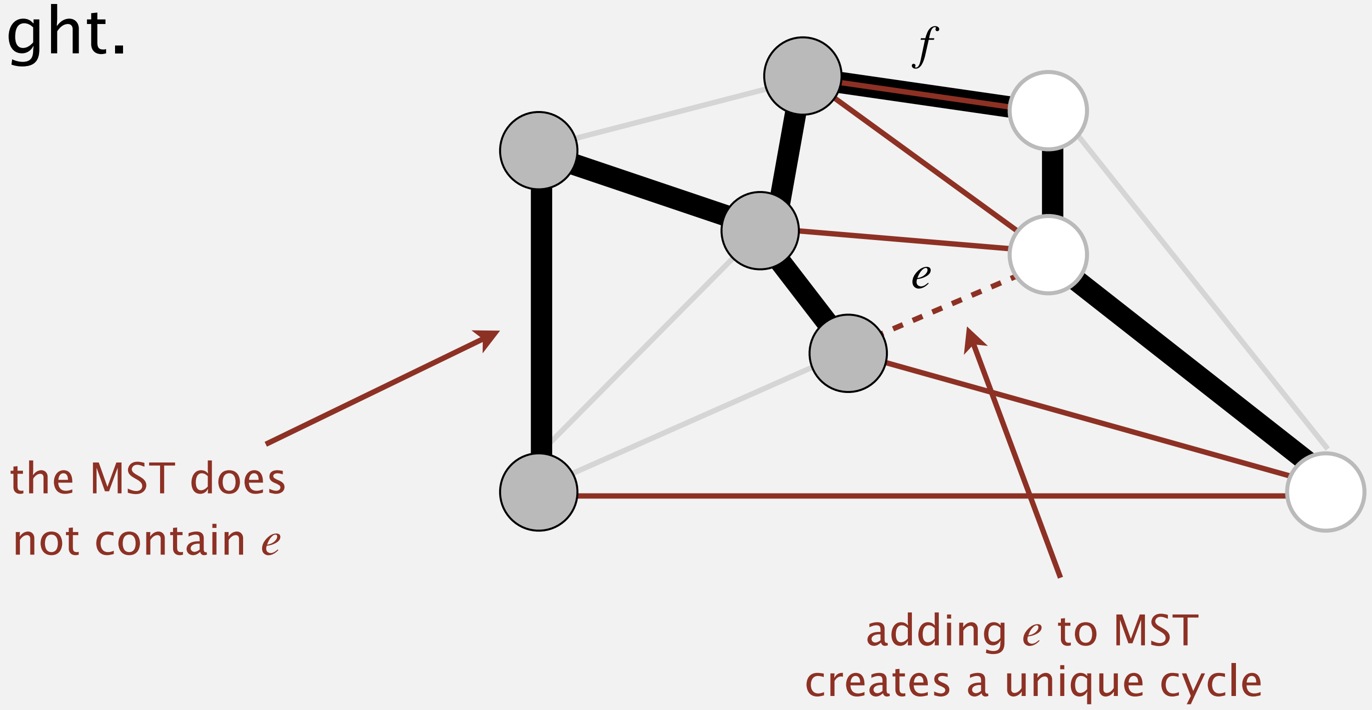
Def. A cut is a partition of a graph's vertices into two (nonempty) sets.

Def. A crossing edge connects two vertices in different sets.

Cut property. Given any cut, the min-weight crossing edge e is in the MST.

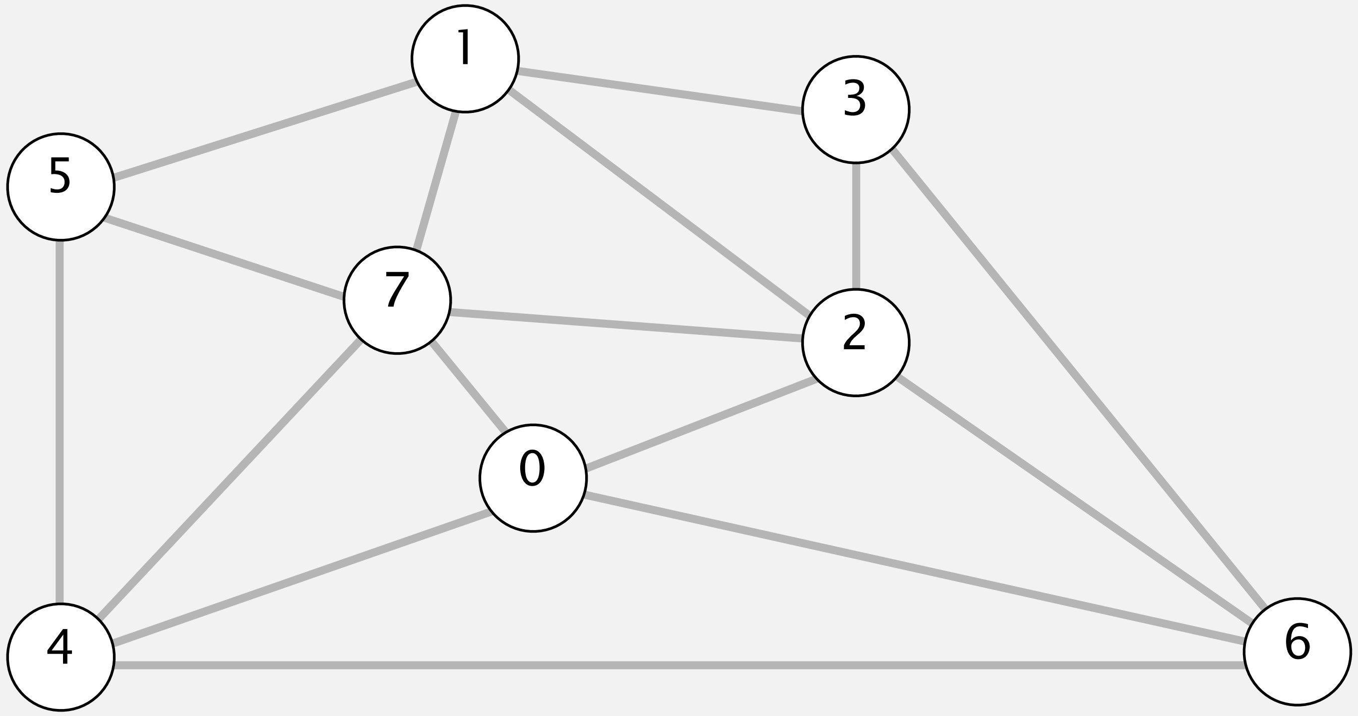
Pf. [by contradiction] Suppose e is not in the MST.

- Adding e to the MST creates a cycle.
- Some other edge f in cycle must be a crossing edge.
- Removing f and adding e is also a spanning tree.
- Since weight of e is less than the weight of f, that spanning tree has lower weight.
- Contradiction. ▪



Greedy MST algorithm demo

- Start with all edges colored gray.
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until $V-1$ edges are colored black.



an edge-weighted graph

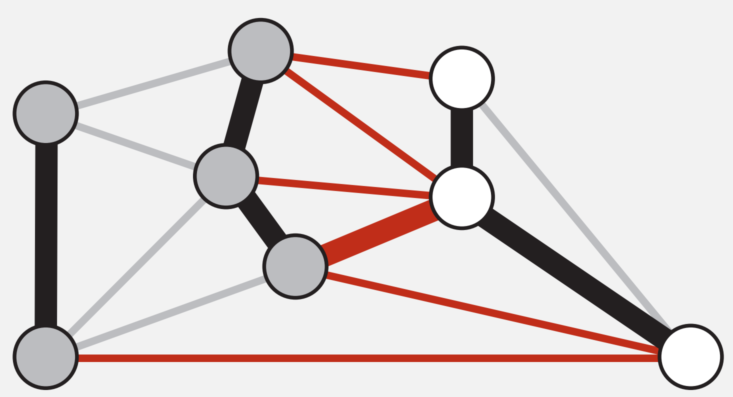
0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Greedy MST algorithm: correctness proof

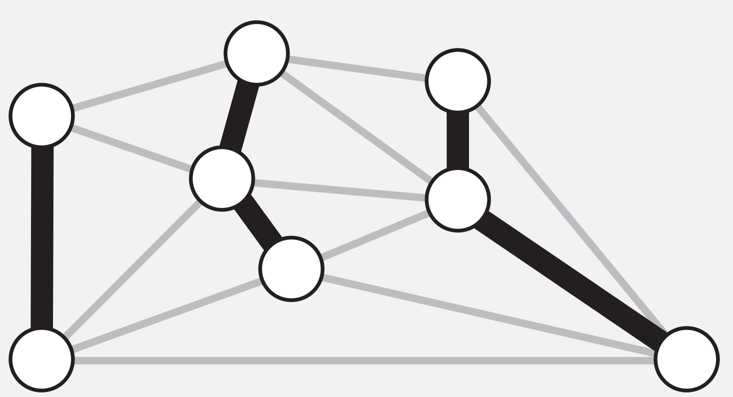
Proposition. The greedy algorithm computes the MST.

Pf.

- Any edge colored black is in the MST (via cut property).
- Fewer than $V-1$ black edges $\Rightarrow$ cut with no black crossing edges. (consider cut whose vertices are any one connected component)



a cut with no black crossing edges



fewer than V–1 edges colored black

Greedy MST algorithm: efficient implementations

Proposition. The greedy algorithm computes the MST.

Efficient implementations. Find cut? Find min-weight edge?

Ex 1. Kruskal's algorithm. [stay tuned]

Ex 2. Prim's algorithm. [stay tuned]

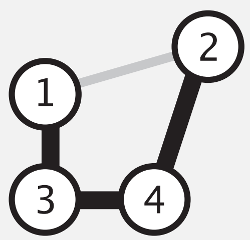
Ex 3. Borüvka's algorithm.

Removing two simplifying assumptions

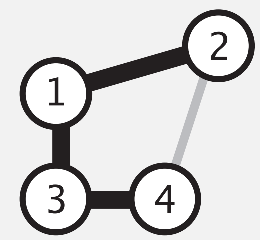
Q. What if edge weights are not all distinct?

A. Greedy MST algorithm still finds a MST!

(our correctness proof fails, but that can be fixed)



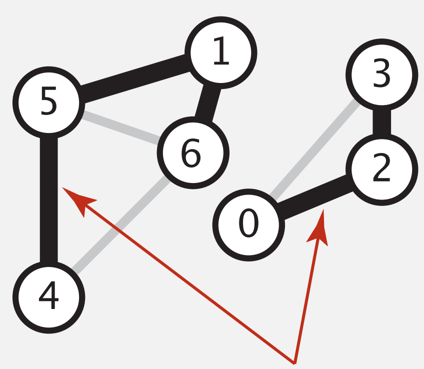
1	2	1.00
1	3	0.50
2	4	1.00
3	4	0.50



1	2	1.00
1	3	0.50
2	4	1.00
3	4	0.50

Q. What if graph is not connected?

A. Finds a minimum spanning forest = MST of each connected component.



4	5	0.61
4	6	0.62
5	6	0.88
1	5	0.11
2	3	0.35
0	3	0.6
1	6	0.10
0	2	0.22

https://algs4.cs.princeton.edu

4.3 Minimum Spanning Trees

- *introduction*
- *cut property*
- *edge-weighted graph API*
- *Kruskal's algorithm*
- *Prim's algorithm*

Weighted edge API

Edge abstraction needed for weighted edges.

```
public class Edge implements Comparable<Edge>
```

	`Edge(int v, int w, double weight)`	*create a weighted edge v-w*
`int`	`either()`	*either endpoint*
`int`	`other(int v)`	*the endpoint that's not v*
`int`	`compareTo(Edge that)`	*compare this edge to that edge*
`double`	`weight()`	*the weight*
`String`	`toString()`	*string representation*

Idiom for processing an edge `e`: `int v = e.either(), w = e.other(v);`

Weighted edge: Java implementation

```
public class Edge implements Comparable<Edge>
{
   private final int v, w;
   private final double weight;

   public Edge(int v, int w, double weight)
   {
      this.v = v;
      this.w = w;
      this.weight = weight;
   }

   public int either()
   {  return v;  }

   public int other(int vertex)
   {
      if (vertex == v) return w;
      else return v;
   }

   public int compareTo(Edge that)
   {
      if      (this.weight < that.weight) return -1;
      else if (this.weight > that.weight) return +1;
      else                                return  0;
   }
}
```

constructor

either endpoint

other endpoint

compare edges by weight

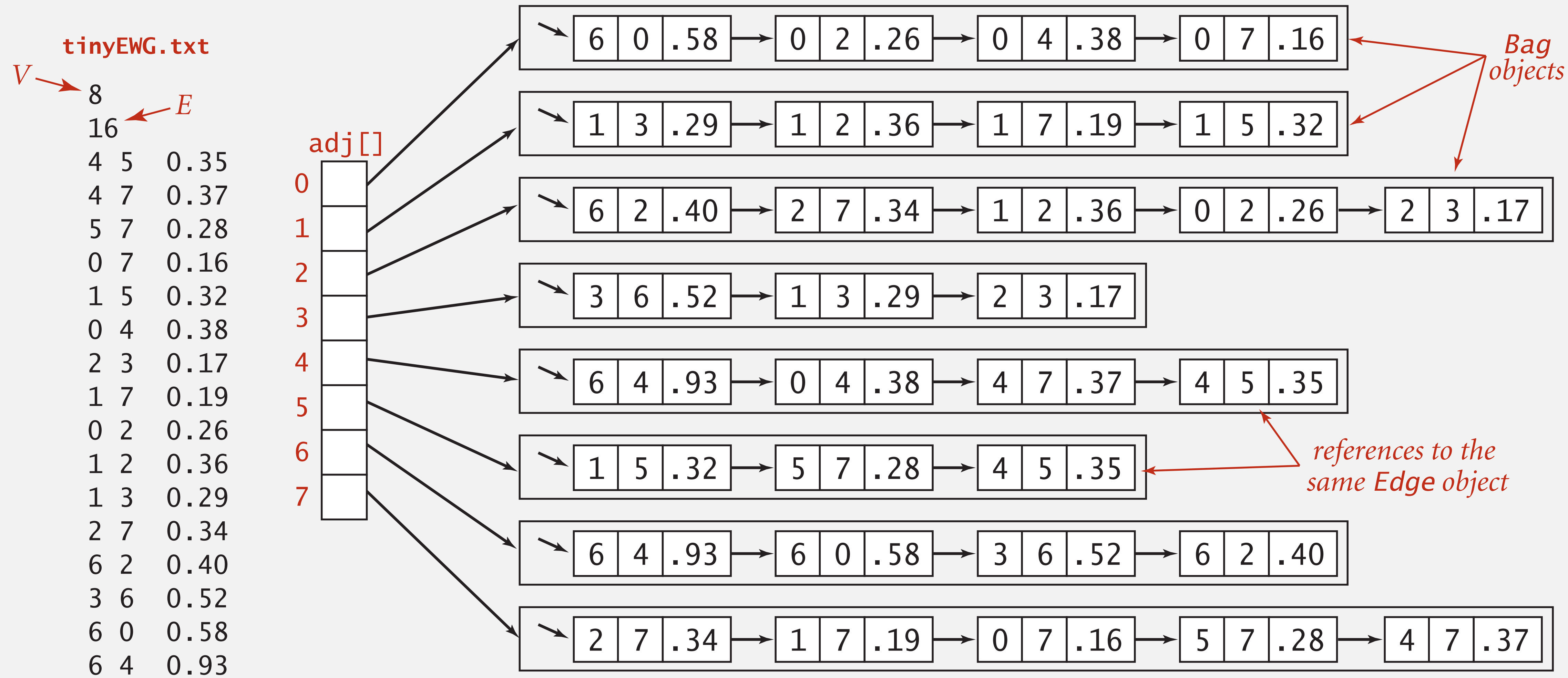
Edge-weighted graph API

public class	EdgeWeightedGraph	
	EdgeWeightedGraph(int V)	*create an empty graph with V vertices*
void	addEdge(Edge e)	*add weighted edge e to this graph*
Iterable<Edge>	adj(int v)	*edges incident to v*
Iterable<Edge>	edges()	*all edges in this graph*
int	V()	*number of vertices*
int	E()	*number of edges*

Conventions. Allow self-loops and parallel edges.

Edge-weighted graph: adjacency-lists representation

Maintain vertex-indexed array of Edge lists.



Edge-weighted graph: adjacency-lists implementation

```
public class EdgeWeightedGraph
{
   private final int V;
   private final Bag<Edge>[] adj;

   public EdgeWeightedGraph(int V)
   {
     this.V = V;
     adj = (Bag<Edge>[]) new Bag[V];
     for (int v = 0; v < V; v++)
        adj[v] = new Bag<Edge>();
   }

   public void addEdge(Edge e)
   {
     int v = e.either(), w = e.other(v);
     adj[v].add(e);
     adj[w].add(e);
   }

   public Iterable<Edge> adj(int v)
   {  return adj[v];  }
}
```

same as Graph, but adjacency lists of Edges instead of integers

constructor

add edge to both adjacency lists

Minimum spanning tree API

Q. How to represent the MST?

public class MST		
	MST(EdgeWeightedGraph G)	*constructor*
Iterable<Edge>	edges()	*edges in MST*
double	weight()	*weight of MST*

https://algs4.cs.princeton.edu

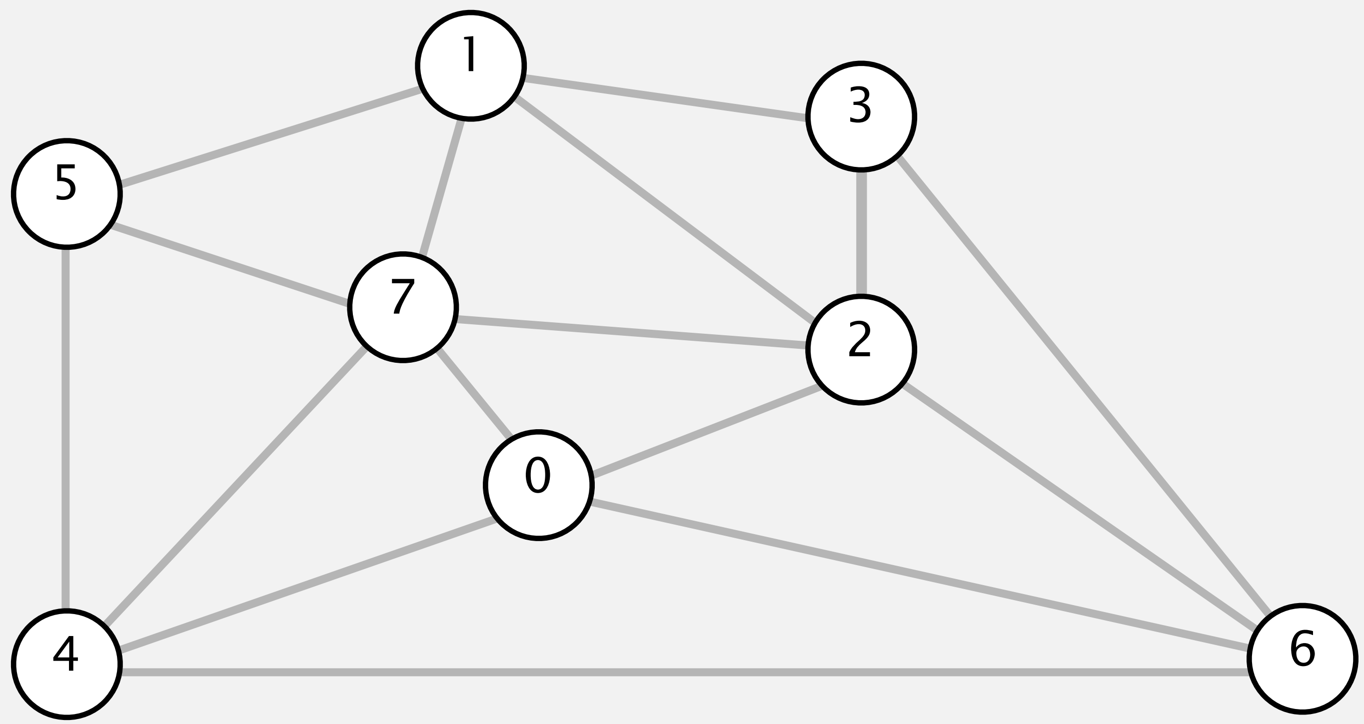
4.3 Minimum Spanning Trees

- *introduction*
- *cut property*
- *edge-weighted graph API*
- ***Kruskal's algorithm***
- *Prim's algorithm*

Kruskal's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to tree T unless doing so would create a cycle.



an edge-weighted graph

graph edges sorted by weight

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

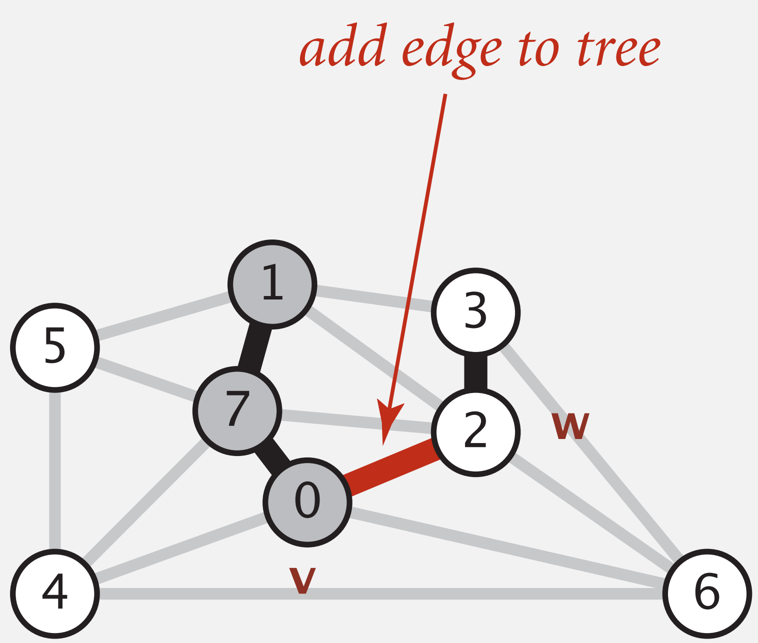
Kruskal's algorithm: visualization

Kruskal's algorithm: correctness proof

Proposition. [Kruskal 1956] Kruskal's algorithm computes the MST.

Pf. [Case 1] Kruskal's algorithm adds edge $e = v\text{–}w$ to T.

- Vertices v and w are in different connected components of T.
- Cut = set of vertices connected to v in T.
- By construction of cut, no edge crossing cut is in T.
- No edge crossing cut has lower weight. Why?
- Cut property $\Rightarrow$ edge e is in the MST.

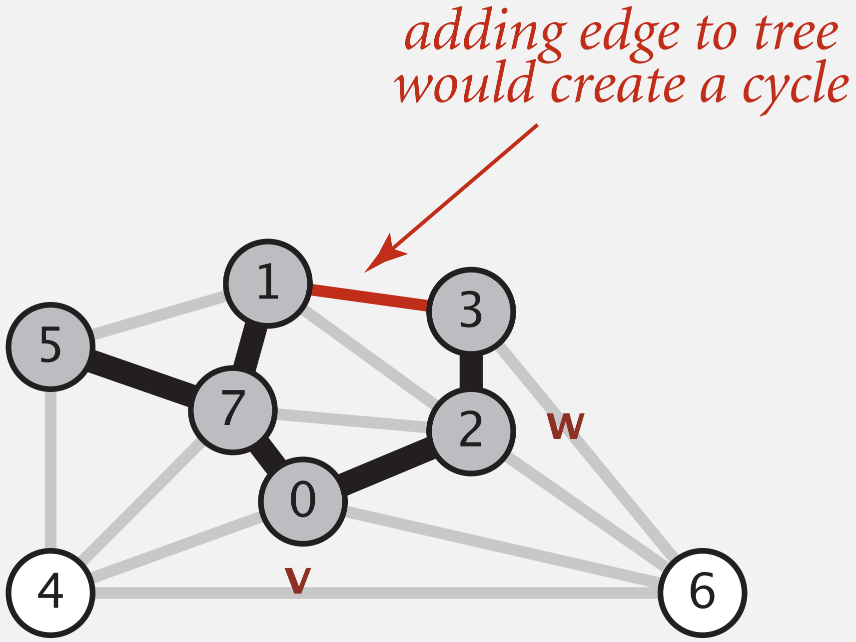


Kruskal's algorithm: correctness proof

Proposition. [Kruskal 1956] Kruskal's algorithm computes the MST.

Pf. [Case 2] Kruskal's algorithm discards edge $e = v–w$.

- From Case 1, all edges in T are in the MST.
- The MST can't contain a cycle. ▪



Minimum spanning trees: quiz 3

Challenge. Would adding edge $v\text{–}w$ to tree T create a cycle? If not, add it.

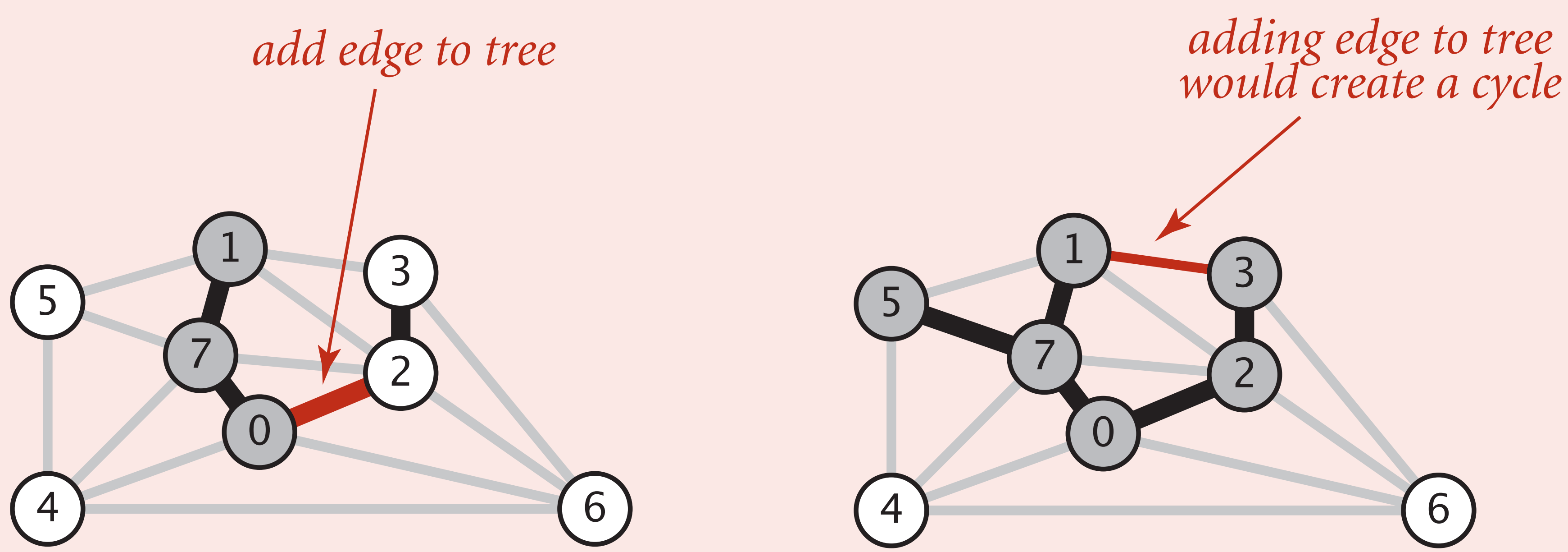
How difficult to implement?

A. 1

B. $\log V$

C. V

D. $E + V$



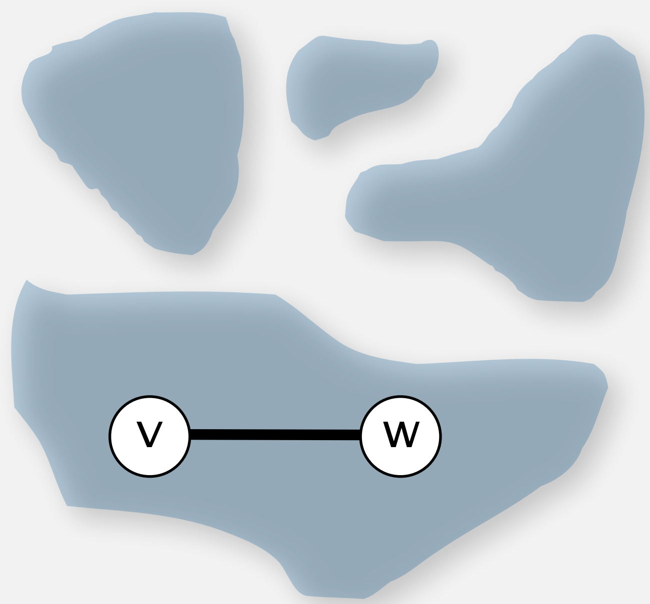
Case 1: v and w in same component
Case 2: v and w in different components

Kruskal's algorithm: implementation challenge

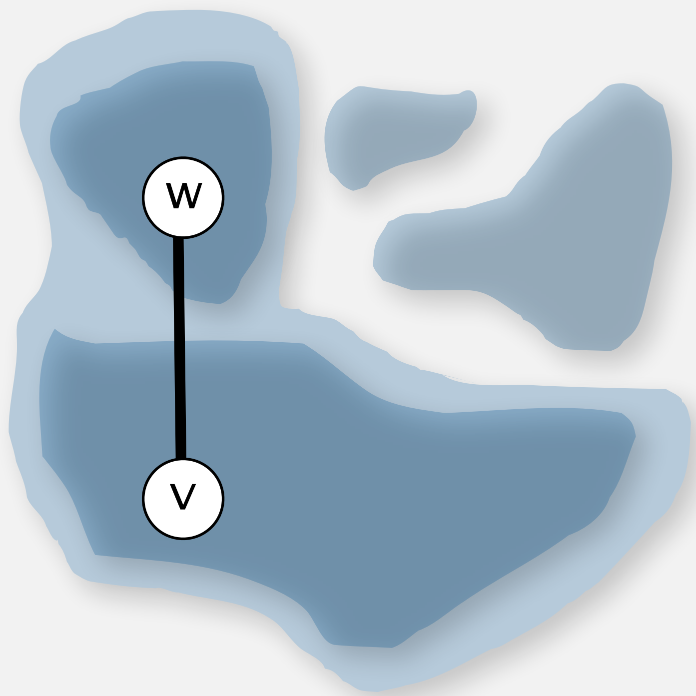
Challenge. Would adding edge $v\text{–}w$ to tree T create a cycle? If not, add it.

Efficient solution. Use the union–find data structure.

- Maintain a set for each connected component in T.
- If v and w are in same set, then adding $v\text{–}w$ would create a cycle.
- To add $v\text{–}w$ to T, merge sets containing v and w.



Case 2: adding v–w creates a cycle



Case 1: add v–w to T and merge sets containing v and w

Kruskal's algorithm: Java implementation

```
public class KruskalMST
{
   private Queue<Edge> mst = new Queue<Edge>();

   public KruskalMST(EdgeWeightedGraph G)
   {
      DirectedEdge[] edges = G.edges();
      Arrays.sort(edges);
      UF uf = new UF(G.V());

      for (int i = 0; i < G.E(); i++)
      {
         Edge e = edges[i];
         int v = e.either(), w = e.other(v);
         if (uf.find(v) != uf.find(w))
         {
            uf.union(v, w);
            mst.enqueue(e);
         }
      }
   }

   public Iterable<Edge> edges()
   {  return mst;  }
}
```

- edges in the MST
- sort edges by weight
- maintain connected components
- greedily add edges to MST
- edge v–w does not create cycle
- merge connected components
- add edge e to MST

Kruskal's algorithm: running time

Proposition. Kruskal's algorithm computes MST in time proportional to $E \log V$ (in the worst case).

Pf.

operation	frequency	time per op
SORT	1	$E \log E$
UNION	$V - 1$	$\log V$ †
FIND	$2 E$	$\log V$ †

same as $E \log V$ if no parallel edges (note on the SORT time per op)

† using weighted quick union

Greed is good

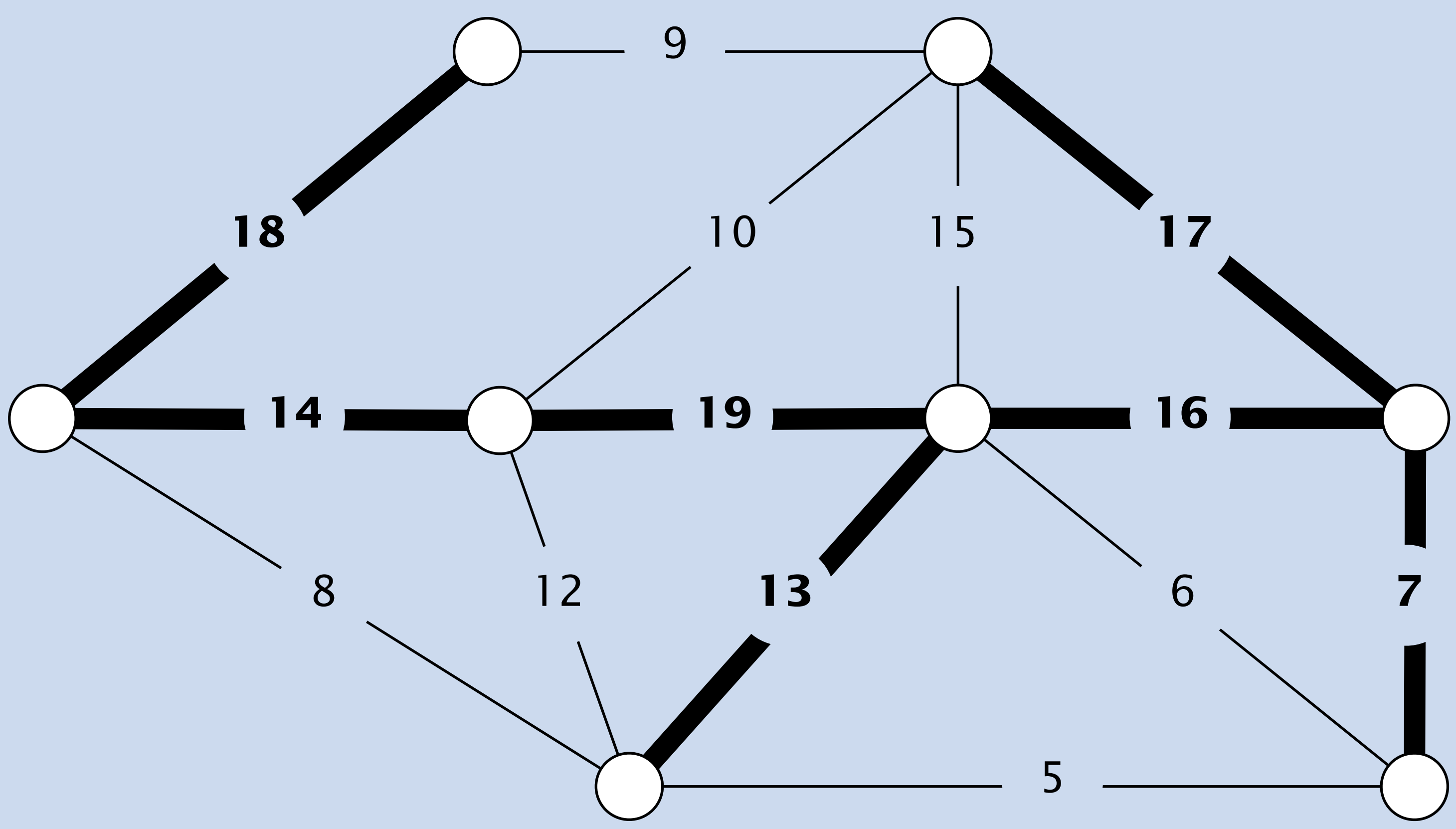
Gordon Gecko (Michael Douglas) evangelizing the importance of greed (in algorithm design?)

Wall Street (1986)

Maximum Spanning Tree

Problem. Given an undirected graph G with positive edge weights, find a spanning tree that maximizes the sum of the edge weights.

Running time. $E \log E$ (or better).



maximum spanning tree T (weight = 104)

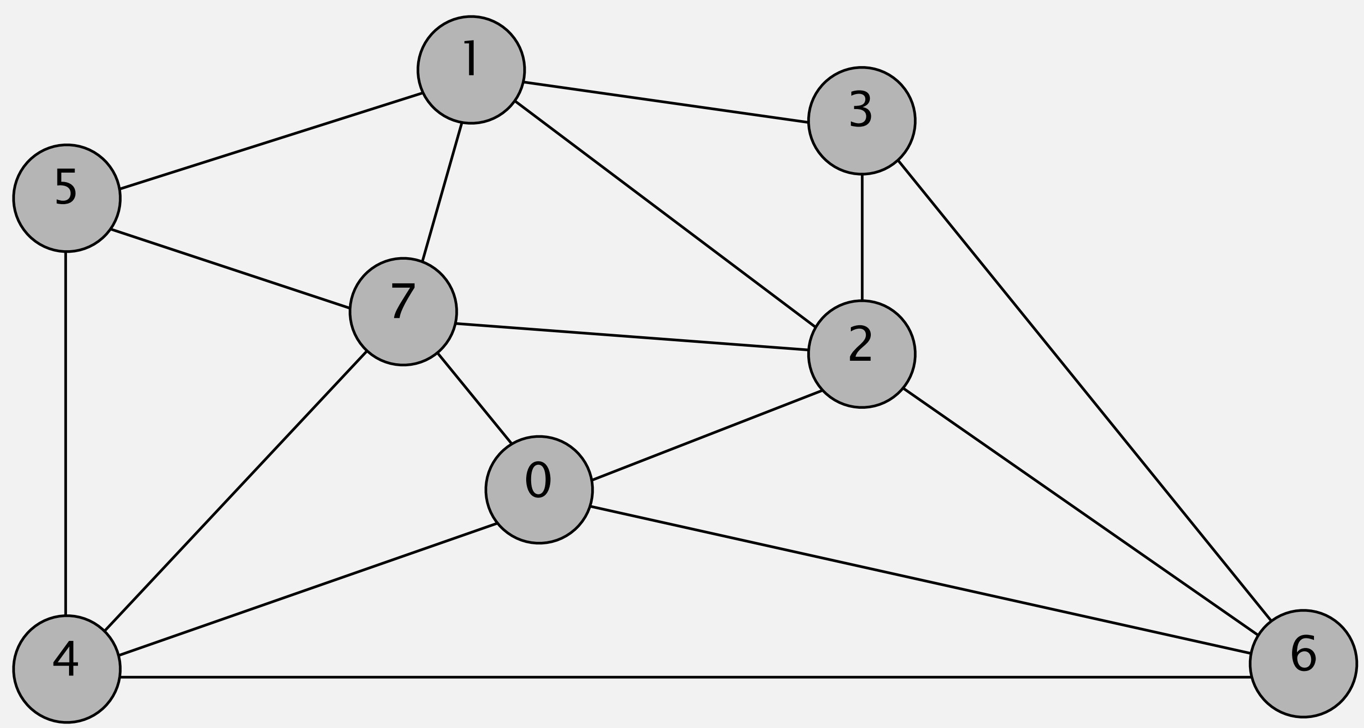
https://algs4.cs.princeton.edu

4.3 Minimum Spanning Trees

- *introduction*
- *cut property*
- *edge-weighted graph API*
- *Kruskal's algorithm*
- *Prim's algorithm*

Prim's algorithm demo

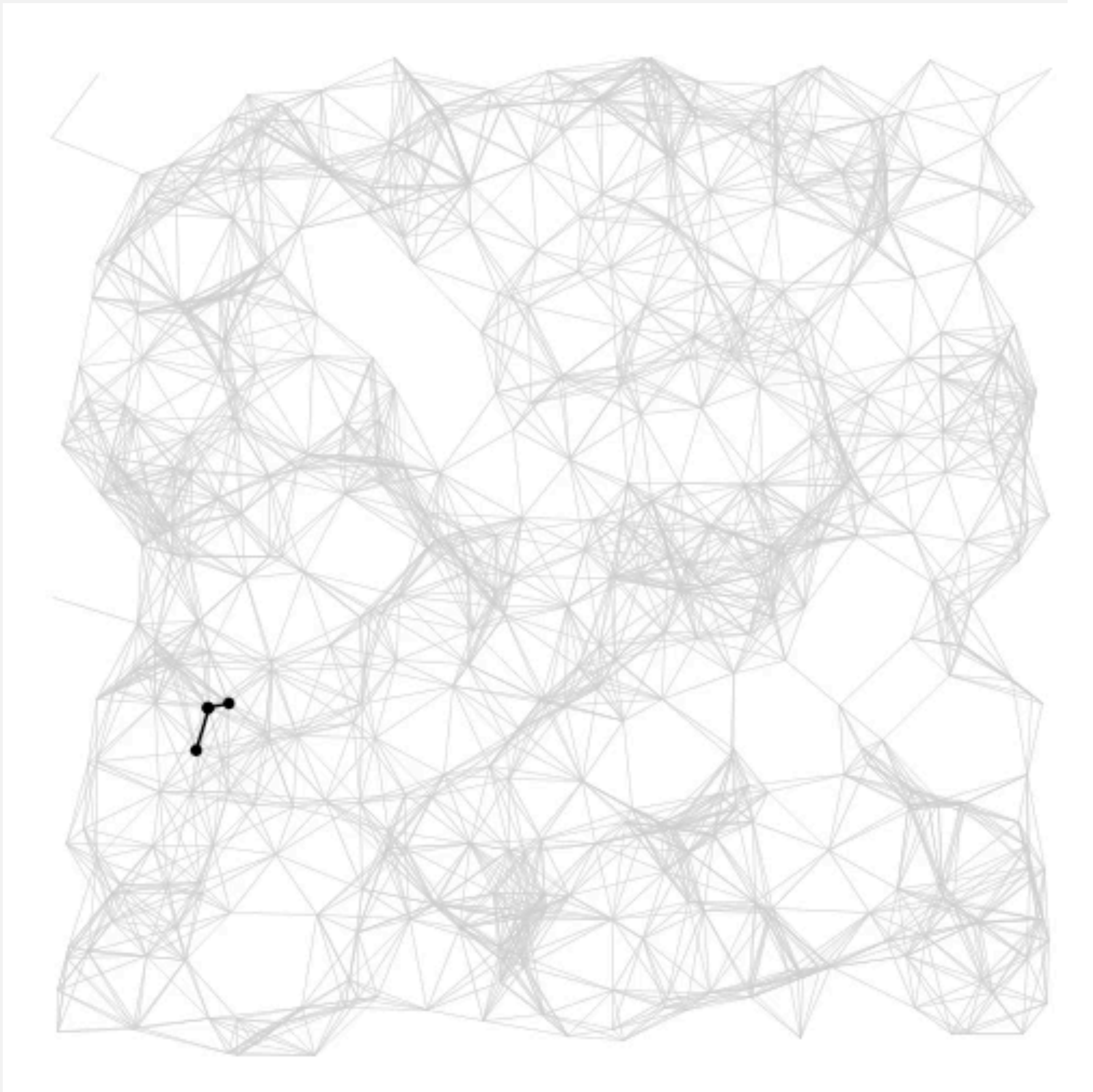
- Start with vertex 0 and greedily grow tree T.
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until $V - 1$ edges.



an edge-weighted graph

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Prim's algorithm: visualization

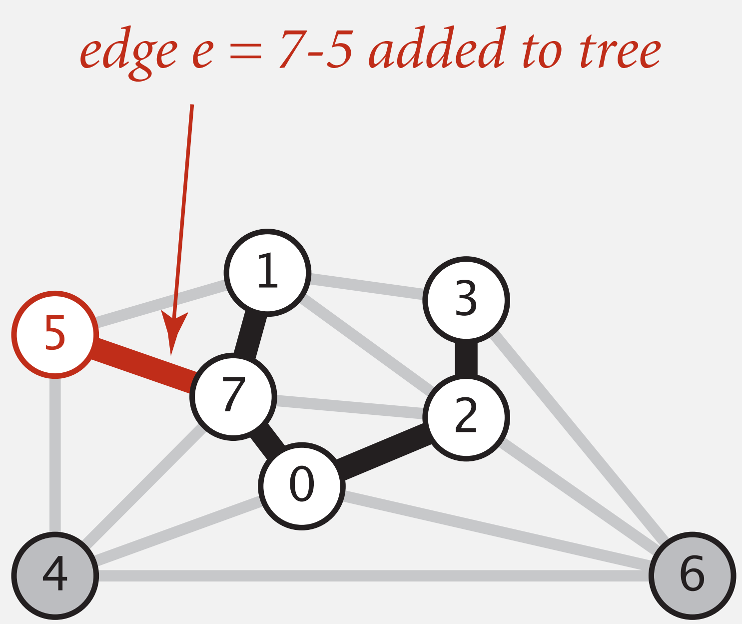


Prim's algorithm: proof of correctness

Proposition. [Jarník 1930, Dijkstra 1957, Prim 1959]
Prim's algorithm computes the MST.

Pf. Let e = min weight edge with exactly one endpoint in T.

- Cut = set of vertices in T.
- No crossing edge is in T.
- No crossing edge has lower weight.
- Cut property $\Rightarrow$ edge e is in the MST. ▪



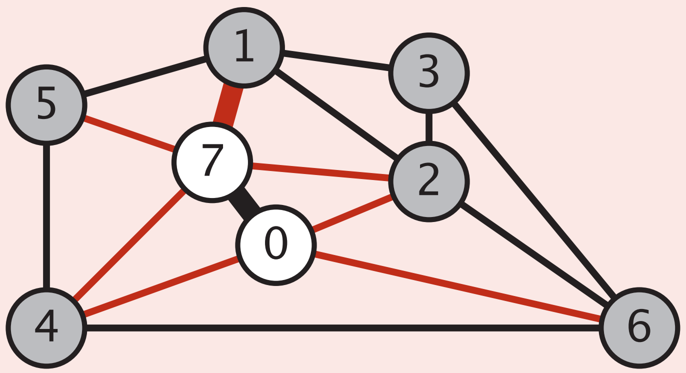
Minimum spanning trees: quiz 4

Challenge. Find the min weight edge with exactly one endpoint in T.

How difficult to implement?

- **A.** 1
- **B.** $\log E$
- **C.** V
- **D.** E

1-7 is min weight edge with exactly one endpoint in T



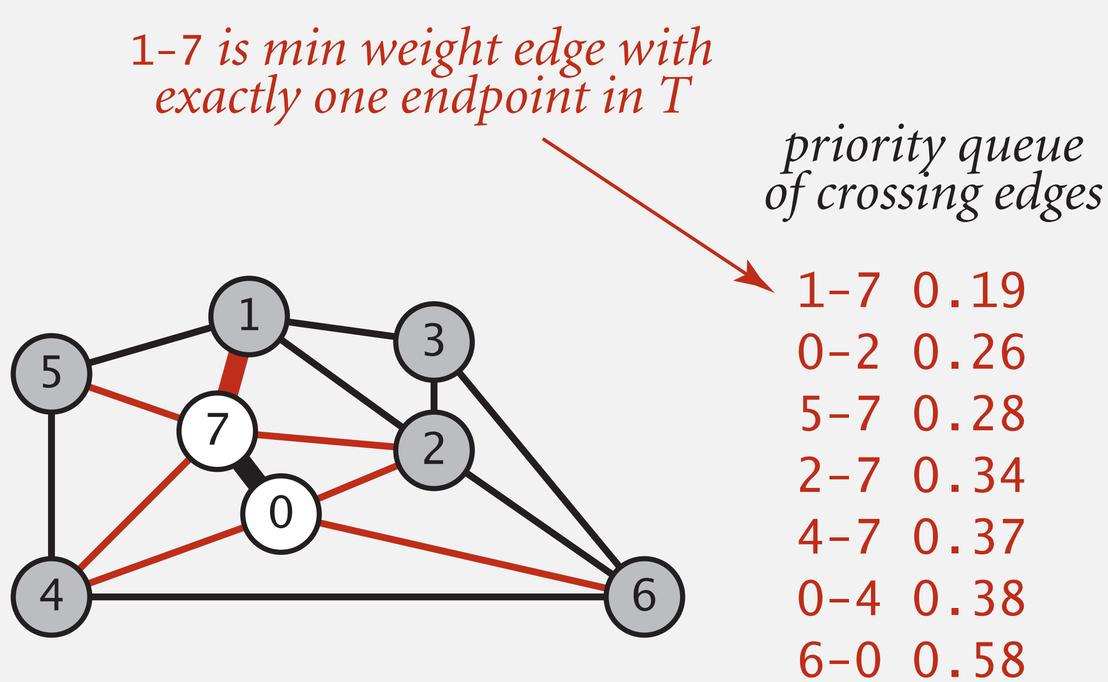

```
1-7 0.19
0-2 0.26
5-7 0.28
2-7 0.34
4-7 0.37
0-4 0.38
6-0 0.58
```

Prim's algorithm: lazy implementation

Challenge. Find the min weight edge with exactly one endpoint in T.

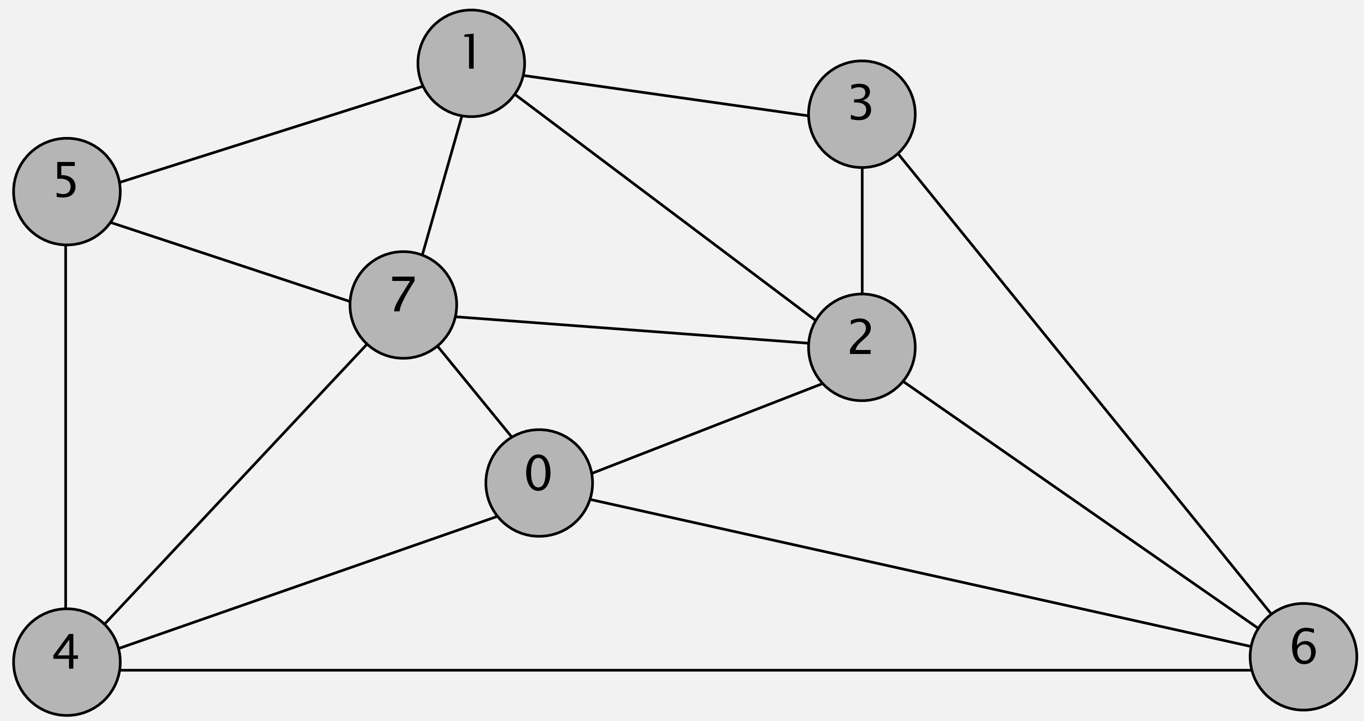
Lazy solution. Maintain a PQ of edges with (at least) one endpoint in T.

- Key = edge; priority = weight of edge.
- DELETE-MIN to determine next edge $e = v–w$ to add to T.
- If both endpoints v and w are marked (both in T), disregard.
- Otherwise, let w be the unmarked vertex (not in T):
  - add e to T and mark w
  - add to PQ any edge incident to w (assuming other endpoint not in T)



Prim's algorithm: lazy implementation demo

- Start with vertex 0 and greedily grow tree T.
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until $V - 1$ edges.



an edge-weighted graph

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Prim's algorithm: lazy implementation

```
public class LazyPrimMST
{
   private boolean[] marked;   // MST vertices
   private Queue<Edge> mst;    // MST edges
   private MinPQ<Edge> pq;     // PQ of edges

    public LazyPrimMST(WeightedGraph G)
    {
        pq = new MinPQ<Edge>();
        mst = new Queue<Edge>();
        marked = new boolean[G.V()];
        visit(G, 0);

        while (!pq.isEmpty() && mst.size() < G.V() - 1)
        {
           Edge e = pq.delMin();
           int v = e.either(), w = e.other(v);
           if (marked[v] && marked[w]) continue;
           mst.enqueue(e);
           if (!marked[v]) visit(G, v);
           if (!marked[w]) visit(G, w);
        }

   }
}
```

- `visit(G, 0);` — assume G is connected
- `Edge e = pq.delMin();` — repeatedly delete the min weight edge e = v–w from PQ
- `if (marked[v] && marked[w]) continue;` — ignore if both endpoints in T
- `mst.enqueue(e);` — add edge e to tree
- `if (!marked[v]) visit(G, v);` / `if (!marked[w]) visit(G, w);` — add either v or w to tree

Prim's algorithm: lazy implementation

```
private void visit(WeightedGraph G, int v)
{
  marked[v] = true;
  for (Edge e : G.adj(v))
     if (!marked[e.other(v)])
        pq.insert(e);
}

public Iterable<Edge> mst()
{  return mst;  }
```

add v to T

for each edge e = v–w, add to PQ if w not already in T

Lazy Prim's algorithm: running time

Proposition. Lazy Prim's algorithm computes the MST in time proportional to $E \log E$ and extra space proportional to E (in the worst case).

minor defect

Pf.

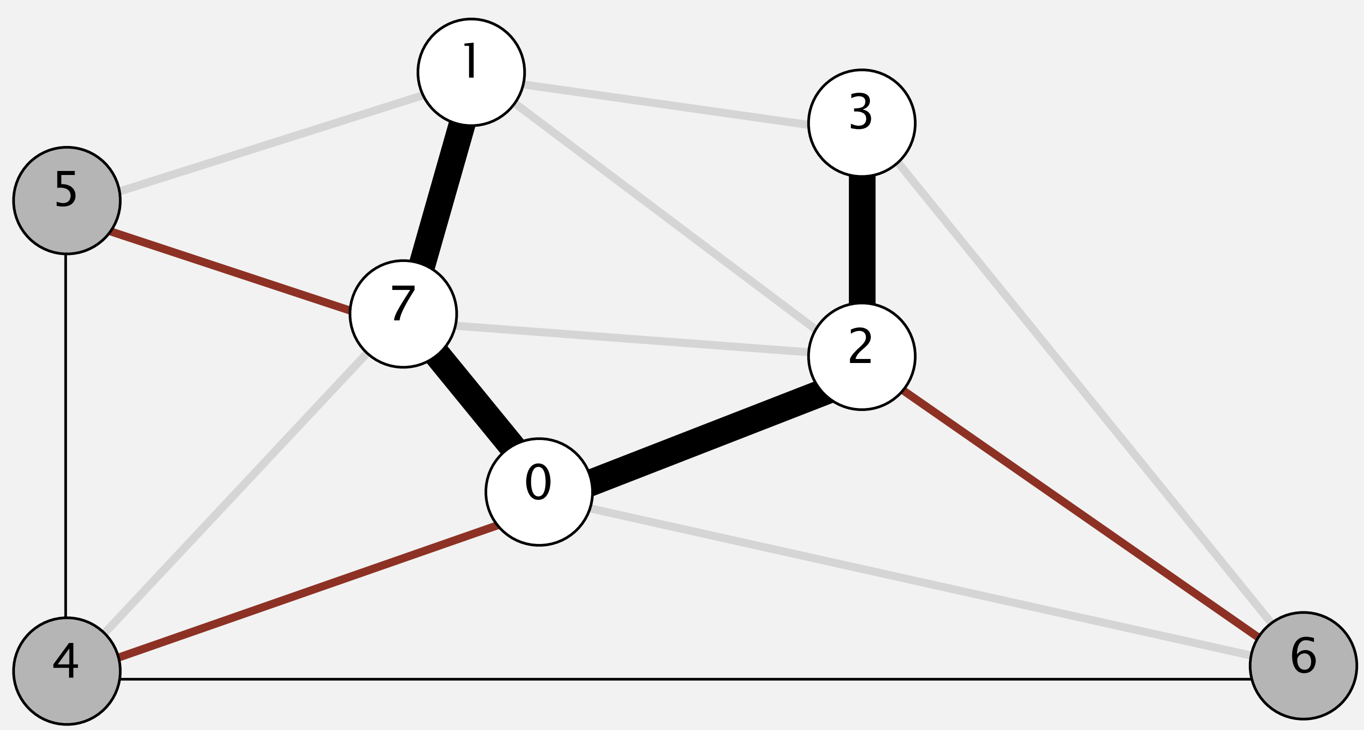
operation	frequency	binary heap
DELETE-MIN	E	$\log E$
INSERT	E	$\log E$

Prim's algorithm: eager implementation

Challenge. Find min weight edge with exactly one endpoint in T.

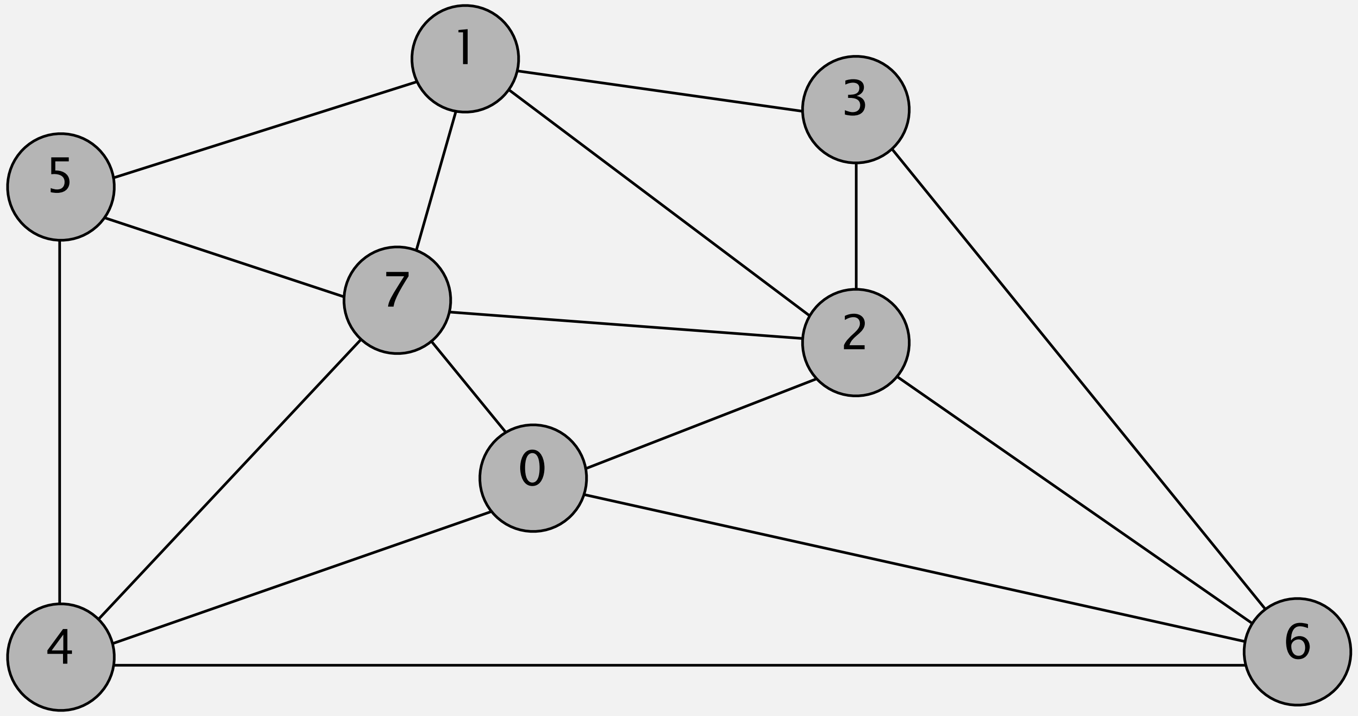
Observation. For each vertex v, need only lightest edge connecting v to T.

- MST includes at most one edge connecting v to T. Why?
- If MST includes such an edge, it must take lightest such edge. Why?



Prim's algorithm: eager implementation demo

- Start with vertex 0 and greedily grow tree T.
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until $V-1$ edges.

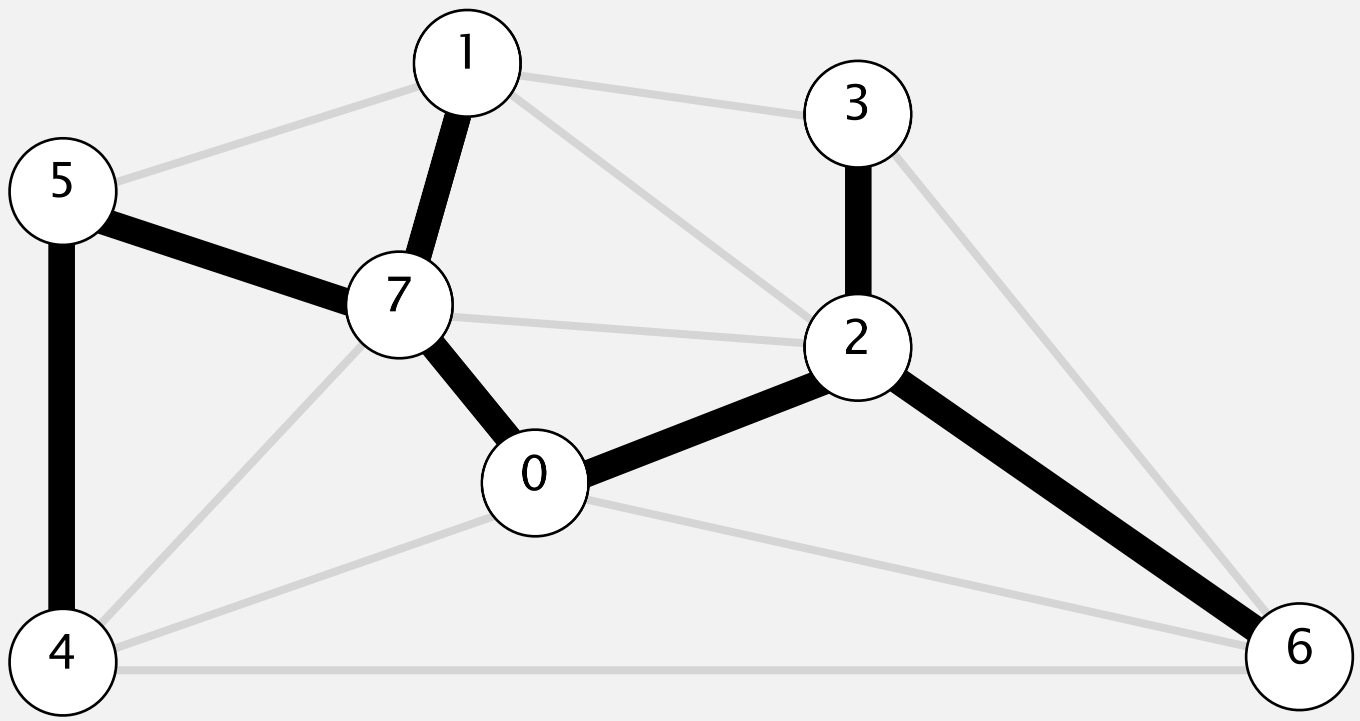


an edge-weighted graph

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Prim's algorithm: eager implementation demo

- Start with vertex 0 and greedily grow tree T.
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until $V - 1$ edges.



v	edgeTo[]	distTo[]
0	-	-
7	0–7	0.16
1	1–7	0.19
2	0–2	0.26
3	2–3	0.17
5	5–7	0.28
4	4–5	0.35
6	6–2	0.40

MST edges

0-7 1-7 0-2 2-3 5-7 4-5 6-2

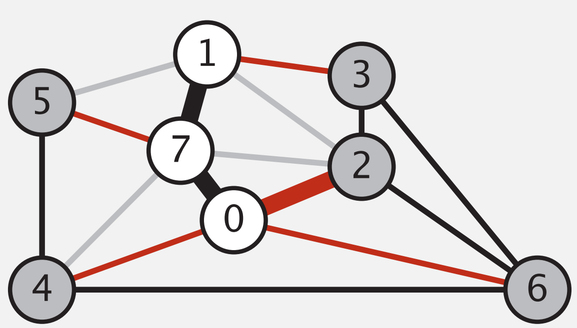
Prim's algorithm: eager implementation

Challenge. Find min weight edge with exactly one endpoint in T.

PQ has at most one entry per vertex

Eager solution. Maintain a PQ of vertices connected by an edge to T, where priority of vertex v = weight of lightest edge connecting v to T.

- Delete min vertex v; add its associated edge $e = v\text{–}w$ to T.
- Update PQ by considering all edges $e = v\text{–}x$ incident to v
  - ignore if x is already in T
  - add x to PQ if not already on it
  - decrease priority of x if $v\text{–}x$ becomes lightest edge connecting x to T



0		
1	1-7	0.19
2	0-2	0.26
3	1-3	0.29
4	0-4	0.38
5	5-7	0.28
6	6-0	0.58
7	0-7	0.16

red: on PQ

black: on MST

Indexed priority queue

Associate an index between 0 and $n-1$ with each key in a priority queue.

- Insert a key associated with a given index.
- Delete a minimum key and return associated index.
- Decrease the key associated with a given index.

for Prim's algorithm: $n = V$, index = vertex, key = weight

```
public class IndexMinPQ<Key extends Comparable<Key>>
```

	`IndexMinPQ(int n)`	*create indexed PQ with indices* $0, 1, \ldots, n-1$
`void`	`insert(int i, Key key)`	*associate key with index i*
`int`	`delMin()`	*remove a minimal key and return its associated index*
`void`	`decreaseKey(int i, Key key)`	*decrease the key associated with index i*
`boolean`	`contains(int i)`	*is i an index on the priority queue?*
`boolean`	`isEmpty()`	*is the priority queue empty?*
`int`	`size()`	*number of keys in the priority queue*

Indexed priority queue: implementation

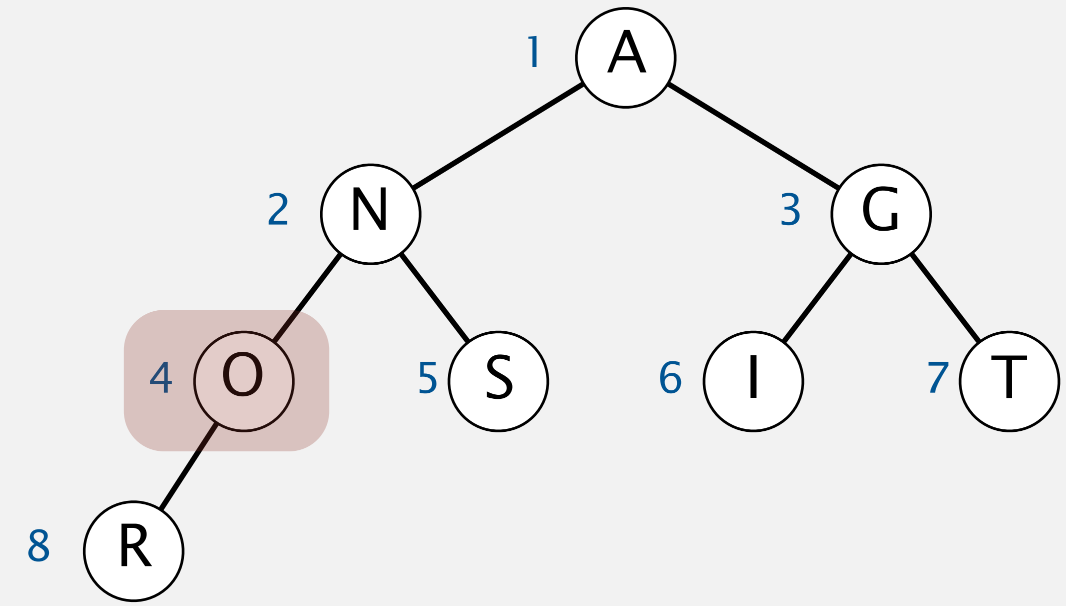
Binary heap implementation. [see Section 2.4 of textbook]

- Start with same code as `MinPQ`.
- Maintain parallel arrays so that:
  - `keys[i]` is the priority of vertex `i`
  - `qp[i]` is the heap position of vertex `i`
  - `pq[i]` is the index of the key in heap position `i`
- Use `swim(qp[i])` to implement `decreaseKey(i, key)`.

i	0	1	2	3	4	5	6	7	8
keys[i]	A	S	O	R	T	I	N	G	-
qp[i]	1	5	4	8	7	6	2	3	-
pq[i]	-	0	6	7	2	1	5	4	3

vertex 2 is at heap index 4

decrease key of vertex 2 to C



Prim's algorithm: which priority queue?

Depends on PQ implementation: V INSERT, V DELETE-MIN, $\leq E$ DECREASE-KEY.

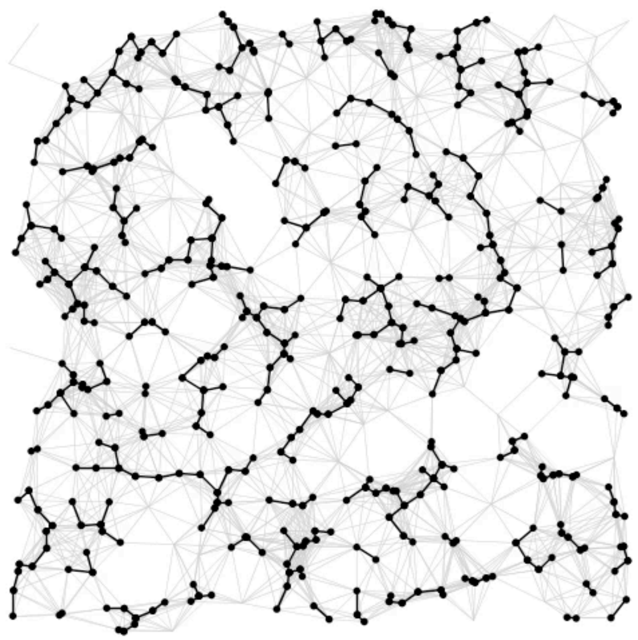
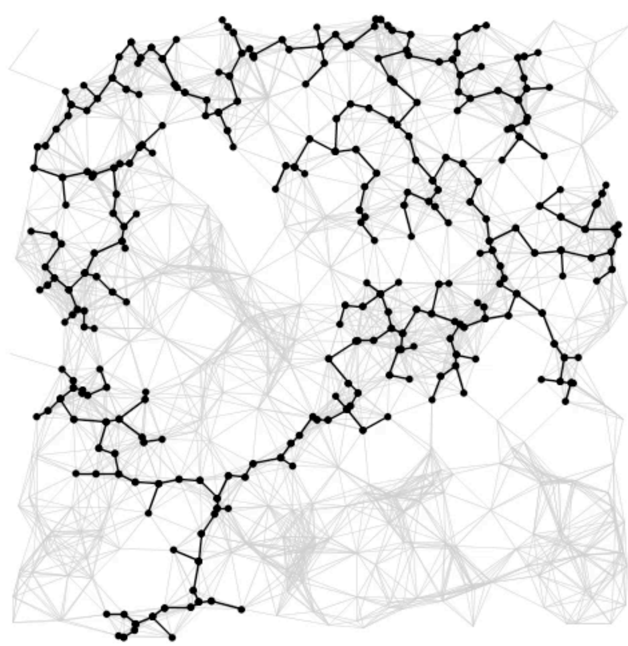
PQ implementation	INSERT	DELETE-MIN	DECREASE-KEY	total
unordered array	1	V	1	V^2
binary heap	$\log V$	$\log V$	$\log V$	$E \log V$
d-way heap	$\log_d V$	$d \log_d V$	$\log_d V$	$E \log_{E/V} V$
Fibonacci heap	1 †	$\log V$ †	1 †	$E + V \log V$

† amortized

Bottom line.

- Array implementation optimal for complete graphs.
- Binary heap much faster for sparse graphs.
- 4-way heap worth the trouble in performance-critical situations.
- Fibonacci heap best in theory, but not worth implementing.

MST: algorithms of the day

algorithm	visualization	bottleneck	running time
Kruskal		sorting union–find	$E \log V$
Prim		priority queue	$E \log V$